NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FEDERATED BAYESIAN NEURAL NETWORKS FOR PASSIVE SONAR

by

Justin M. Loomis

March 2023

Thesis Advisor:	Marko Orescanin
Second Reader:	Armon C. Barton

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2023	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
FEDERATED BAYESIAN NEURAL NETWORKS FOR PASSIVE SONAR

5. FUNDING NUMBERS

6. AUTHOR(S) Justin M. Loomis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Although federated learning and Bayesian neural networks have been researched, there are few implementations of the federated learning of Bayesian networks. In this thesis, a federated learning training environment for Bayesian neural networks using a public code base, Flower, is developed. With it is the exploration of state-of-the-art architecture, residual networks, and Bayesian versions of it. These architectures are then tested with independently and identically distributed (IID) datasets and non-IID datasets derived from the Dirichlet distribution. Results show that the MC Dropout version of Bayesian neural networks can achieve state-of-the-art results—91% accuracy—for IID partitions of the CIFAR10 dataset through federated learning. When the partitions are non-IID, federated learning through inverse variance aggregation of probabilistic weights does as well as its deterministic counterpart, with roughly 83% accuracy. This shows that Bayesian neural networks can be federated and achieve state-of-the-art results as well.

14. SUBJECT TERMS
Bayesian neural networks, federated learning, independently and identically distributed, IID

15. NUMBER OF PAGES
71

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

FEDERATED BAYESIAN NEURAL NETWORKS FOR PASSIVE SONAR

Justin M. Loomis
Lieutenant, United States Navy
BS, United States Naval Academy, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
March 2023**

Approved by: Marko Orescanin
Advisor

Armon C. Barton
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Although federated learning and Bayesian neural networks have been researched, there are few implementations of the federated learning of Bayesian networks. In this thesis, a federated learning training environment for Bayesian neural networks using a public code base, Flower, is developed. With it is the exploration of state-of-the-art architecture, residual networks, and Bayesian versions of it. These architectures are then tested with independently and identically distributed (IID) datasets and non-IID datasets derived from the Dirichlet distribution. Results show that the MC Dropout version of Bayesian neural networks can achieve state-of-the-art results—91% accuracy—for IID partitions of the CIFAR10 dataset through federated learning. When the partitions are non-IID, federated learning through inverse variance aggregation of probabilistic weights does as well as its deterministic counterpart, with roughly 83% accuracy. This shows that Bayesian neural networks can be federated and achieve state-of-the-art results as well.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AI	artificial intelligence
CNN	convolutional neural network
DoD	Department of Defense
FedAvg	federated averaging
FL	federated learning
FTL	federated transfer learning
FedIV	federated with inverse variance weighting
HFL	horizontal federated learning
IID	independently and identically distributed
KLD	Kullback-Leibler divergence
ML	machine learning
MAP	maximum a posteriori
MLE	maximum likelihood estimation
MC	Monte Carlo
NLL	negative log likelihood
NN	neural network
ResNet	residual network
VI	variational inference
VFL	vertical federated learning

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

1.1 Overview

Artificial intelligence (AI) has propelled traditional machine learning (ML) into new fields once thought too complex or computationally expensive. This newfound capability rests on the evolution of deep neural network (NN)s. NN are now able to accomplish complex human tasks in computer vision such as classification, object detection, and image segmentation. In 2016, AlphaGo defeated the top human Go players, a feat that was thought to be outside the realm of computers. Now, AI and deep NNs are being used in every industry such as medicine, finance, automotive industries, retail, advertising, and by the Department of Defense (DoD). Another aspect of AI is deploying it to the edge. For the DoD, deploying AI and ML models at the edge provides a huge benefit. One possible use case is passive sonar analysis. Passive sonar analysis has been a human-only task, but developments in AI and ML could increase performance and speed comparable to the developments in computer vision. Passive sonar, however, can come with a lot of uncertainty due to ever-changing sound propagation profiles and geographic locations, and knowing the uncertainty of a model's prediction is important. Bayesian NNs have the potential to achieve high performance in analyzing passive sonar while providing key insights on their own uncertainty. When paired with an operator, an AI-human team could increase performance in both speed and accuracy for passive sonar analysis. When it comes to deploying these models for passive sonar analysis, having the devices on the edge is an advantage. It reduces communication overhead and increases the speed of inference.

However, there are inherent difficulties with deploying AI on edge devices. First is that AI and ML models deployed on the edge are generally inference only. These models are trained in a separate location and then deployed to the edge device. This introduces the difficulty that each edge device may be exposed to different types of situations and datasets. To overcome this, it would require many diverse locations to collaborate and share their data in order to use AI on the edge, which may or may not be feasible due to the operating environment or classification. Another option would be for edge devices to conduct training with their own dataset. However, this presents a scenario where an edge device could learn vital properties but would be unable to share those properties with other edge devices. The edge devices would be unable to contribute their own inferences and datasets to other devices. The communication, collaboration, and time required to improve edge-AI devices with the traditional AI and ML pipeline are significant. Despite these issues, there are increased difficulties in creating large datasets with the addition of privacy laws such as GDPR EU, CCPA, and China's Cyber Security Law [1]. This problem is exacerbated with the DoD for datasets that are classified. If the trend continues throughout the globe, datasets are more likely to be distributed and isolated. One example of this is in the medical field. Hospitals cannot share information about their patients with other hospitals. Each hospital is a data silo separated from the rest. In the sense of the DoD, collaborating on a large dataset between different branches and agencies is an intimidating task and difficult to coordinate at the speed necessary to compete with other nations, such as China. In addition to difficulties within the DoD, as the nation begins to rely more on partners and allies, the DoD is going to have to work on various information-sharing agreements. The datasets that the DoD uses to train its AI may not be readily shareable with allies and partners.

To address these issues of deploying devices to the edge and data privacy, the Navy and DoD should use federated learning (FL). It would be further advantageous to use Bayesian NNs in a FL setting. While it may be time and computationally expensive to train models on the edge with a large dataset, FL removes the need for each site to contain a large dataset. Sites with a small subset of the global dataset can conduct FL and perform nearly as well as a centrally trained model with a large dataset, improve generality across the sites, and reduce communication costs.

1.2 Considerations for the Navy

Using FL is an opportunity to employ AI at the edge and reduce the need to collect a large dataset. This would immensely assist the Navy's efforts in deploying and training AI models in the fleet. Through the traditional AI pipeline, for example, creating an AI model for the Navy would require each command to collaborate in creating a global dataset, whether for passive sonar or network traffic analysis, maintenance, or human resources. It is a highly expensive and time-consuming task that could be obsolete by the time it's completed with the advent of new data. However, FL provides a method for each command to train and deploy a model on their local, current data and aggregate their models with another command's AI models. The communication cost is also minimal since only the model's weights are passed rather than the entire dataset. While FL presents a method to deploy and train AI models on the edge, a Bayesian NN is a model that can provide not only predictions but an estimate of uncertainty to its evaluation. Warfighters operate in uncertain environments and knowing when a deployed AI model is uncertain of its predictions prevents overconfidence for both the AI and the warfighter. This feature can significantly help AI-warfighter teams operate at a higher level of efficiency. Combining these two aspects of the distributed and continual learning properties of FL and the uncertainty of Bayesian NNs would be a huge advantage for the Navy in various applications, such as network traffic analysis, synthetic aperture radar or drone imagery analysis, or passive sonar analysis.

1.3 Research Objectives and Contributions

To demonstrate this, I developed a FL framework for comparing Bayesian NNs to their deterministic counterparts and analyzed their results in this thesis. The main contribution of this thesis is this framework is benchmarked on a known dataset, CIFAR10 [2], to compare results. The dataset is well-studied in FL research [3]–[6]. The AI model architecture used is the residual network (ResNet) [7]. It is a state-of-the-art neural network architecture and sets a baseline for the CIFAR10 dataset. This allows Bayesian ResNets to be compared to the original's state-of-the-art results in both a centralized and FL setting. The main questions that this thesis intends to answer are the following:

- How to aggregate Bayesian NNs?
- How does FL impact the performance of a Bayesian NN?
- Can FL improve a NN's overall performance?
- How does a Bayesian NN compare to a deterministic one in FL?

1.4 Organization

The rest of this thesis is organized into four additional chapters. Chapter 2 introduces ResNets, Bayesian NNs and variational inference (VI), and FL. Chapter 3 discusses the dataset in a centralized and FL training scheme, the ResNet models used, and the various experiments conducted. Chapter 4 examines the experiment's results to understand the impact of FL environments on Bayesian NNs. Chapter 5 presents my findings and suggests future work and considerations going forward.

CHAPTER 2: Background

2.1 Overview of Neural Networks

NNs are the backbone of deep machine learning. With roots inspired by a human neuron, NNs take the concept of having multiple inputs going to a neuron, which will activate if a threshold is met. For a given activation function, z, the neuron activates and produces an output such that $output = z(input * w + b)$, where w is the weight value, and b is the bias. By combining multiple layers of neurons, the model gets "deeper", hence the moniker "deep" learning. NNs became more popular once GPUs were discovered to increase training and inference speeds. This has resulted in significant progress in computer vision, natural language processing, and etc. for AI.

One version of a NN has become famous for computer vision tasks such as object detection, image segmentation, and object classification. This version is called a convolutional neural network (CNN). Instead of using the entire image at once, it convolves a smaller kernel across the image, creating a collection of activations called filters. In this way, the model is able to learn about localized parts of the image. As the NN gets deeper, its receptive field gets larger, meaning that a deeper convolutional layer in a NN is able to see outputs derived from the entire image. Convolutional NNs have seen significant advances in computer vision tasks and the ResNet uses these types of layers.

2.1.1 Overview of ResNet

One of the issues with deep NNs such as the VGG nets [8], is that they contain millions of parameters and require significant computer resources to use. Another issue was that the deeper a NN got, the performance began to degrade [9], [10]. Motivated by this degradation problem, Kaiming He et al. introduced the ResNet [7]. This network increased the depth of the model while being less complex than the VGG net. The ResNet solves the degradation from deep NNs by using a deep residual learning framework. He et al hypothesized that if one considers, $H(x)$ being the mapping of a few stacked layers and x being the initial input, these stacked layers can approximate a residual function. This function can be expressed as $F(x) := H(x) - x$. The original function, $H(x)$, is then represented as $F(x) + x$ [7]. This reformulation was motivated by the degradation problem that He et al. faced with models with increasing depth.

The architecture of a residual network begins with a pair of convolutional layers with a kernel size of 3 x 3. Any downsampling occurs with a stride of 2, halving the feature map. This forms a residual block and follows two design rules:

1. The layers have the same filter size and
2. If the feature map is downsampled, the filter size is doubled.

To turn this architecture into a residual network, identity shortcuts are added to connect the input and output of the residual block. These shortcuts also follow two rules:

1. Adding extra padding to match the increased dimension size or
2. A projection shortcut is used via a convolutional layer with a kernel size of 1 x 1 to match the dimensions.

Figure 2.1 visualizes a residual block that takes input x, the identity, and adds it to the activation of the two convolutional layers. This is based on He et al's hypothesis that it is "easier to optimize the residual mapping than to optimize the original, unreferenced mapping" [7].

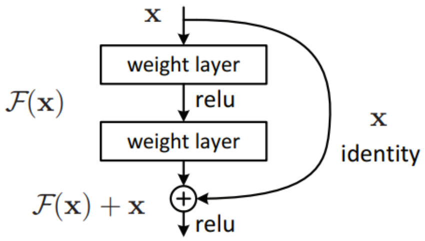

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
x
weight layer
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.1. ResNet Residual Block. Source: [7].

The ResNet was chosen for this research due to its success and documentation, specifically the ResNet with a depth of 20 (ResNet20). The ResNet20 architecture consists of seven 32 x 32 convolutional layers with 16 filters, followed by six 16 x 16 convolutional layers with 32 filters, then six 8 x 8 convolutional layers with 64 filters, ending with an average pooling layer and dense layer with softmax activation. All layers used L2 regularization and in between each residual block was a batch normalization layer. Figure 2.2 gives an example of the model's architecture [7].

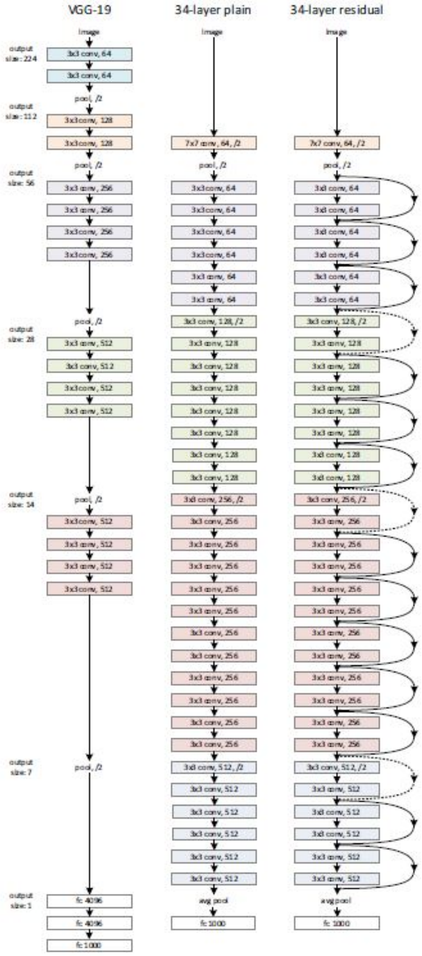


Figure 2.2. ResNet Architecture Example. Source: [7].

2.1.2 Overview of MLE and MAP

In general, the ResNet20, like many AI models, would attempt to model the likelihood given by the probability of getting a label, y, given its features, x, and the model's parameters, θ ($p(y|x, \theta)$). This is known as using maximum likelihood estimation (MLE) because it attempts to maximize the likelihood. Assuming that this probability distribution is a normal one, the log of the likelihood becomes

$$Log - Likelihood = \log \frac{1}{\sqrt{2\pi\sigma^2}} - \log \sigma - \frac{|y - \mu|^2}{2\sigma^2} \tag{2.1}$$

A MLE version of an AI model attempts to maximize the likelihood to get the best prediction given an input's features and the model's parameters. In practice, the likelihood is maximized by taking the negative log likelihood (NLL) and minimizing that value. The Loss function is shown in equation 2.2.

$$Loss = -\log \frac{1}{\sqrt{2\pi\sigma^2}} + \log \sigma + \frac{|y - \mu|^2}{2\sigma^2} \tag{2.2}$$

In equation 2.2, the parameters, θ, seem to disappear. However, μ is a function of x and θ such that $f(x, \theta) = \mu$. During training, the loss of a model is calculated and then its gradient is taken with respect to θ. This can be done by the chain rule since μ is a function of x and θ. For the CIFAR10 dataset, in order to maximize the likelihood, the loss must be calculated with categorical cross-entropy. This allows the gradient to be taken with respect to each of the datasets ten classes. This means that the NLL needs to be calculated across all the classes in the dataset as shown below:

$$Loss = -\sum_{i=1}^{c} p(y = i|x) * \log p(y = i|x, \theta) \tag{2.3}$$

This type of loss, MLE, applied to a ResNet scores nearly 100% on the training data, but only 89% on the test data. This is not ideal and is a sign of overfitting. Overfitting is when the model adapts too well to the training data and loses generality for inputs outside of that subset. While the MLE model was trying to maximize the prediction of y given our features, x, and some model's parameters, θ, it is more effective to maximize the posterior or $p(\theta|D)$ where D is the training data given by $D = (x_0, y_0)...(x_i, y_i)...(x_n, y_n)$ using a method known as maximum a posteriori (MAP) estimation. These models are making predictions based on the parameters derived from the training data and get a point estimation of our posterior. The posterior, $p(\theta|D)$, is summarized as the highest probability where θ is the set of parameters given our data, D. By Bayes Law, the posterior can be broken up into the following equation:

$$p(\theta|D) = \frac{p(D|\theta) * p(\theta)}{p(D)} \tag{2.4}$$

The $p(D|\theta)$ should look familiar because it is the likelihood, $p(y|x, \theta)$. However, it now adds a prior, $p(\theta)$. To use this in a MAP model, the loss is derived by first taking the log of the posterior.

$$\log p(\theta|D) = \log p(D|\theta) + \log p(\theta) - \log p(D) \tag{2.5}$$

This term is to be maximized but in practice, the negative of the $\log p(D|\theta)$ is taken and then minimized as seen below

$$\underset{\theta}{\operatorname{argmin}}(\log p(\theta|D)) = \underset{\theta}{\operatorname{argmin}}(-\log p(D|\theta) - \log p(\theta) + \log p(D)) \tag{2.6}$$

To solve for the prior we use results from Eq. 2.1. The $\log p(D)$ is ignored since we're taking the derivative of the loss concerning θ and the term is dropped. Further, it is assumed that the $p(\theta)$ is of a normal distribution centered over zero. Therefore, if we take the log of the prior it becomes

$$-\log p(\theta) = -\log \frac{1}{\sqrt{2\pi\sigma^2}} + \log \sigma + \frac{|\theta|^2}{2\sigma^2} \tag{2.7}$$

The negative sign is brought down with the prior since in equation 2.6 the log prior is subtracted from the log-likelihood. Also, μ could be removed since we assume the prior is centered over zero. Since the loss is going to be taken with respect to θ, the term $\log p(D)$ is removed and $\frac{1}{2\sigma^2}|\theta|^2$ is leftover. This is calculated over all the weight values and becomes the $\sum_{i=1}^{w} \frac{1}{2\sigma^2}|\theta_i|^2$. The likelihood, $p(D|\theta)$, is taken over all the data points, N, and averaged. The resulting loss equation is

$$\underset{\theta}{\operatorname{argmin}}(-\log p(\theta|D)) = \underset{\theta}{\operatorname{argmin}}(-\frac{1}{N}\sum_{i=1}^{N} \log p(D_i|\theta) - \frac{1}{N}\sum_{j=1}^{w} \frac{1}{2\sigma^2}|\theta_j|^2) \tag{2.8}$$

As discussed in section 2.1.1, my implementation of the ResNet uses L2 regularization. That is to say, the loss function is described in equation 2.8. This reduces the overfitting problem from models using the MLE loss function described in equation 2.2 and produces results as seen in He et al.'s paper [7].

2.2 Overview of Variational Inference

The ResNet that utilizes a MAP loss function can be described as a deterministic model. Meaning that for a given input, the model will always give the same output. This is because MAP seeks to learn a point estimation of the posterior, $p(\theta|D)$. However, a Bayesian NN seeks to learn the distribution of the posterior, a continuous value. Unfortunately, calculating the exact probability distribution of the posterior is an intractable problem in most cases. Usually, approximating the posterior's probability distribution is sufficient and Bayesian models do just that. By approximating a probability distribution instead of point estimation, the model is able to express epistemic uncertainty. This approximation is given as $q(\theta)$. The goal is to have the model learn a $q(\theta)$ that is as close as possible to the posterior, $p(\theta|D)$. To understand the distance between the approximated posterior, $q(\theta)$, and the actual posterior, $p(\theta|D)$, the Kullback-Leibler divergence (KLD) is used. Therefore, the goal for the model is for the KLD between $q(\theta)$ and $p(\theta|D)$ to be as small as possible. In equations 2.9 - 2.19, Bayes theorem defines the posterior in the KLD as the likelihood, prior, and data distributions.

$$KL[q(\theta)||p(\theta|D)] \tag{2.9}$$

By definition of KLD this becomes the expectation for $q(\theta)$ between the log of $q(\theta)$ over $p(\theta|D)$:

$$E_{q(\theta)}[\log \frac{q(\theta)}{p(\theta|D)}] \tag{2.10}$$

Applying Bayes theorem to the definition of the posterior and the rules of log gets the following series:

$$E_{q(\theta)}[\log \frac{q(\theta)}{\frac{p(D|\theta)p(\theta)}{p(D)}}] \tag{2.11}$$

$$E_{q(\theta)}[\log \frac{q(\theta)p(D)}{p(D|\theta)p(\theta)}] \tag{2.12}$$

$$E_{q(\theta)}[\log q(\theta) + \log p(D) - \log p(D|\theta) - \log p(\theta)] \tag{2.13}$$

$$E_{q(\theta)}[-\log p(D|\theta) + \log q(\theta) - \log p(\theta) + \log p(D)] \tag{2.14}$$

$$E_{q(\theta)}[-\log p(D|\theta) + \log \frac{q(\theta)}{p(\theta)} + \log p(D)] \tag{2.15}$$

$$E_{q(\theta)}[-\log p(D|\theta)] + E_{q(\theta)}[\log \frac{q(\theta)}{p(\theta)}] + E_{q(\theta)}[\log p(D)] \tag{2.16}$$

$$E_{q(\theta)}[-\log p(D|\theta)] + KL[q(\theta)||p(\theta)] + E_{q(\theta)}[\log p(D)] \tag{2.17}$$

$$Loss = \underset{\theta}{\operatorname{argmin}}(E_{q(\theta)}[-\log p(D|\theta)] + KL[q(\theta)||p(\theta)]) \tag{2.18}$$

In the above equations, $p(D)$ is removed since it has no impact on the change of θ. Getting the exact distribution of $q(\theta)$ is intractable in most cases. To compensate for that difficulty, Monte Carlo (MC) sampling, of size M, is used to approximate the distribution. This means that the loss of M sample models is taken and then averaged to get the final result.

$$Loss = \underset{\theta}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^{M} [-\log p(D|\theta^i)] + KL[q(\theta^i)||p(\theta^i)]]) \tag{2.19}$$

2.2.1 Monte Carlo Dropout

A method for conducting variational inference is through MC Dropout [11]. This is done by taking a MAP model and adding dropout layers throughout the architecture. These layers would probably drop a neuron for the preceding layer. Therefore, dropout at inference allows multiple models to be sampled via MC sampling. Due to the dropout layers, the approximation function for the posterior assumes that it is either sampling a normal Gaussian distribution, $N(0, \sigma^2)$, or a Gaussian distribution over the weight, $N(w, \sigma^2)$ as seen in equation 2.20. In practice, however, σ is considered to be so insignificantly small that the means of these distributions, 0 or w, are used during sampling.

$$q(\theta) = p_{drop}N(0, \sigma^2) + 1 - p_{drop}N(w, \sigma^2) \tag{2.20}$$

2.2.2 Variational Inferencing with Flipout

Another method of conducting variational inference is through flipout. In a Flipout model, the model approximates a Gaussian distribution.

$$q(\theta) = N(\mu, \sigma^2) \tag{2.21}$$

During inference $\theta_i = \mu_i + \sigma_i * \epsilon_i * r$, where $\epsilon \sim N(0, 1)$ and $r = [-1, 1]$. A single flipout model can assume various configurations based on r without increasing the parameters. Therefore the loss function for a flipout model remains the same as equation 2.19.

2.3 Overview of Federated Learning

FL has been a rising area of research due to AI being deployed on the edge and increased data regulations [1]. McMahan et al. and Konecny et al. have explored using federated learning for updating language models on mobile phone devices [3], [4], [12]. In terms of data, mobile phones use private data to enable their keyboard's autocorrect functionality. Another use of federated learning was by Sarma et al. [13] for analyzing clinical prostate imaging across three different sites without sharing any of the patient's data. Their research found that FL improved overall accuracy for three different hospital sites than if they were trained on their own datasets separately. Already, FL has a practical application today. According to Yang et al., federated learning is an algorithmic framework that has the following four characteristics:

- Two or more parties want to jointly build a neural network.
- The data does not leave either party during training a model.
- The model can be transferred in a manner that either party cannot reverse-engineer the training data.
- The performance of the model results in an approximation of the ideal model if it was trained centrally with all the data. [1]

In addition to these observations, federated learning systems generally perform less than their centrally trained counterparts [1], [3], [4]. However, the additional data security and privacy guarantees outweigh that potential performance loss [1], [13].

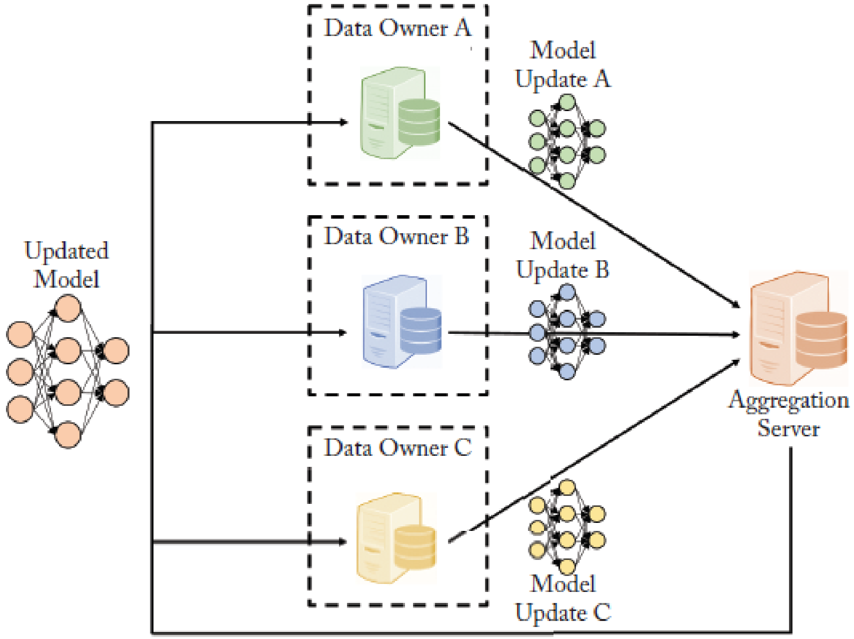


Figure 2.3. Client-Server Federated Learning. Source: [1].

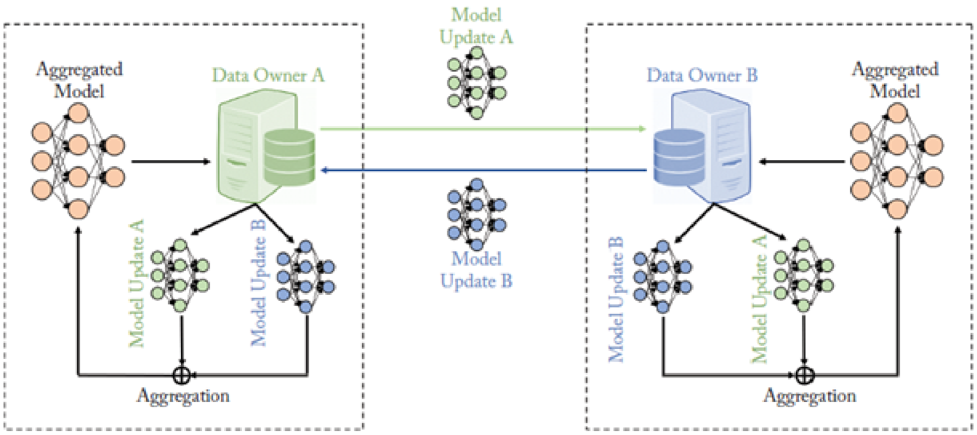


Figure 2.4. Peer-to-Peer Federated Learning. Source: [1].

The usual federated learning concept is a client-server where the server provides the global model and aggregates the client models. As seen in Figure 2.3, the clients receive and train the global model on their local datasets. They then send their updates or weights back to the server. Another type of network is the peer-to-peer network as seen in Figure 2.4 where the global model is updated without a coordinating server. The goal is to improve a model than what an individual site can achieve alone while not sharing any data between participants. Either due to privacy and security concerns or because of communication overhead, FL becomes a viable solution for deploying AI on distributed and embedded hardware. Figure 2.3 shows an example of a client-server model. Notice that each data owner updates the global model with their data, and then transmits the updates to a trusted, coordinating server that then aggregates the updates and distributes the new global model back to each data owner. In Figure 2.4, a peer-to-peer network is represented. The key difference is that the data owners trade updates with each other without a coordinating server. Figure 2.5 demonstrates that this type of model updating can happen with K trainers in either a cyclic transfer or random transfer, also known as Gossip Learning [1], [14].

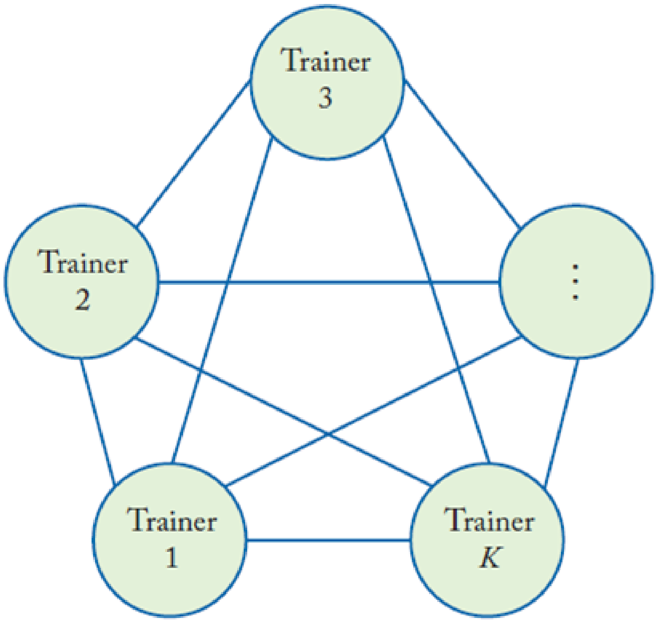


Figure 2.5. Peer-to-Peer Federated Learning with K Trainers. Source: [1].

Besides the network architecture, federated learning can be divided into three different categories:

- horizontal federated learning (HFL)
- vertical federated learning (VFL)
- federated transfer learning (FTL)

Each type is influenced by the dataset's feature space and inference goals of the participants in the federation scheme. Horizontal learning is when each federation site observes the same feature space but has a different subsection of labels. It is known as a sample-based partition of the total dataset [1]. One example is Google Keyboard, where each user and their phone acts as a site observing the same or similar feature space but with different labels. Figure 2.6 demonstrates how two entities can coordinate in a federated learning environment to increase the sample space that the neural network is exposed to.

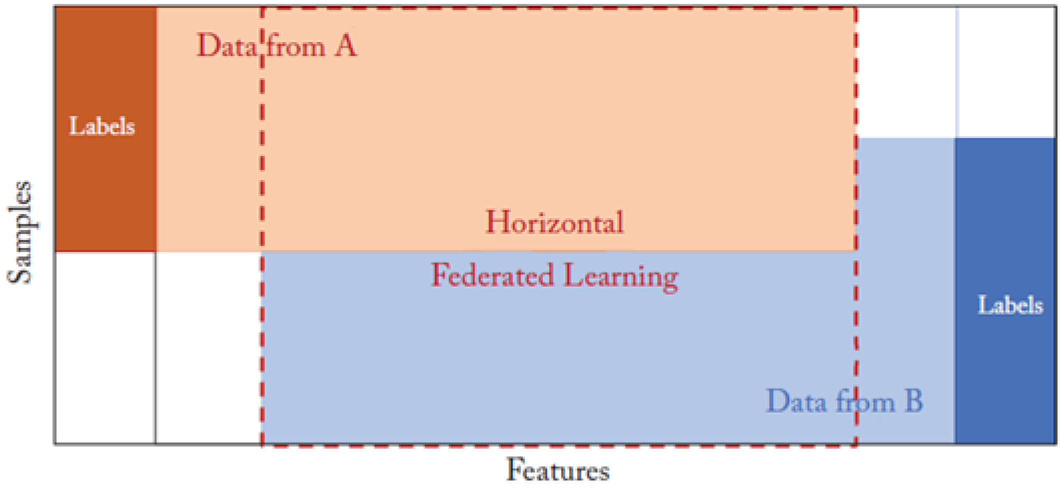


Figure 2.6. Horizontal Federated Learning Example. Source: [1].

Theoretically, it is the same as expanding a database without sharing the data. VFL is when participants of a federation are observing different feature spaces of the same sample. It is known as a feature-based partition of the total dataset [1]. This type of federated learning can be used when two entities are providing different services but share the same sample space, such as a bank and an e-commerce company [1]. Figure 2.7 displays how two entities share the same sample space, but have different feature sets. In cases where horizontal or vertical federated learning is not practical due to highly heterogeneous data, federated transfer learning can be applied to fill in the gaps between participating parties. It attempts to build neural networks for a resource-scarce domain by exploiting knowledge from a resource-rich domain. It is generally divided up into three categories: instance-based, feature-based, and model-based [15].

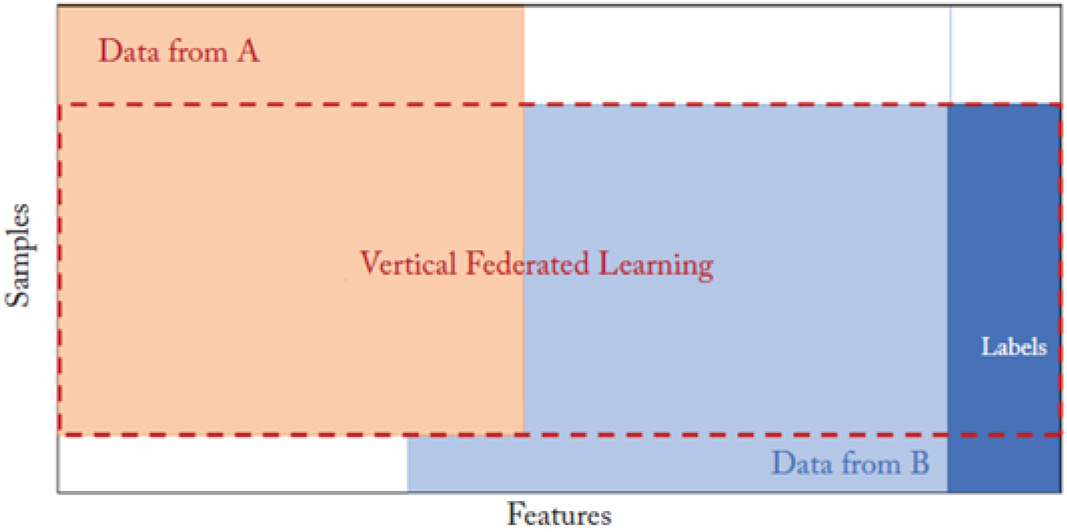


Figure 2.7. Vertical Federated Learning Example. Source: [1].

I explore the use of HFL in a client-server setting for Bayesian NNs. This will be done by dividing the dataset into independently and identically distributed (IID) partitions that will be trained by a local site. Each site will communicate its model's weights back to the server which will then aggregate the updates and redistribute them back to the sites. One of the foundational aggregation methods would be FedAvg [12]. FedAvg takes a weighted average of the sites' updates based on their sample number. Instead of taking the weighted average of the gradient, the weighted average of the weights is calculated as the new NN parameters. Given U users, each one has a subset of the data, D_u^t out of the total D^t. By taking the weighted average of each user's weights, θ, the following equation is given:

$$\theta^{t+1} = \frac{1}{|D^t|} \sum_{u=0}^{U} \delta_u^{t+1} \tag{2.22}$$

In equation 2.22, the global NN parameters after a federation round, θ^{t+1}, is equal to the sum of the site parameters, δ_u^{t+1}, where $\delta_u^{t+1} = |D_u^t|\theta_u^{t+1}$. This type of federated learning reduces the communication overhead required for its gradient-averaging counterpart. However, while federated learning via gradients guarantees a global model's convergence, federating across the site's weights does not [1].

This method of aggregation has primarily been used for NNs with deterministic weights, but there's no consensus on the proper way of aggregating probabilistic weights. I also explore the aggregation of Bayesian models with probabilistic weights which are really aggregating a mean and standard deviation. For instance, in order to take the weighted average of a series of Gaussian probability distributions, $\theta_u^t = N(\mu_u, \sigma_u^2)$ and the average of Gaussian's is given by $N(\mu^t + 1, V^t + 1) = \frac{1}{|D^t|} \sum_{u=0}^{U} |D_u^t| N(\mu_u^{t+1}, V_u^{t+1})$. I use this method and refer to it as federated averaging (FedAvg) as well. I do test another averaging formula as well, inverse variance weighted average [16]. In addition to using the number of examples a site has, it weights the parameters by their inverse variance. The mean and variance are calculated by the following equations:

$$\bar{\mu} = \frac{\sum_i \frac{\mu_i}{\sigma_i^2}}{\sum_i \frac{1}{\sigma_i^2}} \tag{2.23}$$

$$\bar{V} = \frac{1}{\sum_i \frac{1}{\sigma_i^2}} \tag{2.24}$$

The inverse variance weighting in a FL scheme will be referenced as federated with inverse variance weighting (FedIV).

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

This chapter discusses how I explored the concept of implementing FL for Bayesian NNs. First, the original purpose of this thesis was to use FL on edge devices, but due to hardware and time complications, a simulation framework was used instead. The experiment process of comparing Bayesian and deterministic versions of the ResNet20 in both a centralized and FL training scheme is also discussed. These experiments strived to achieve the results of Kaiming He et al's ResNet20 accuracy results, roughly 91% [7]. Next, Bayesian versions of the ResNet20 will be established using MC Dropout and Flipout as discussed in Chapter 2. Once the centralized baseline is established, four different FL experiments will be conducted. Two will be conducted for IID data partitions but will change the number of local epochs, and the last two will be for non-IID data partitions of different degrees.

3.1 Framing the Problem

Originally, the goal of my thesis was to implement Bayesian NNs in a FL setting for passive sonar and to analyze the results. Previously, experiments have been done on FL for edge devices but not for Bayesian NNs [17]. This presented some unknowns as Bayesian NNs have probabilistic weights and a standard averaging may not provide the required results or even be mathematically accurate. Another issue was choosing a framework to conduct FL. At first NVIDIA's NVFLARE python package was investigated for my thesis. With it, I was able to aggregate a MC Dropout Bayesian NN with three clients and one server. Unfortunately, running a ResNet20 required an entire GPU, and I could only afford to have three clients. NVFLARE did not have a simulation function to expand the number of clients and most FL research papers have tens or hundreds of clients involved. Then I discovered the Flower python package which did FL like NVFLARE, between a server and client, and supported embedded development and deployment of AI systems. The key factor was that it also had a simulation function to expand the number of clients with limited resources. I was able to replicate my results with Flower and expand it for Flipout Bayesian NNs too. The next step was conducting full experiments using Flower's simulation function. Ten clients were chosen to demonstrate my experiments.

3.2 CIFAR10 Dataset

Next, it helps to understand the dataset used for the benchmark. The CIFAR10 dataset consists of 60,000 images of 10 classes. Figure 3.1 shows what the 10 classes are and some sample images. This dataset has been used as the problem set and baseline for many papers [2], [7], [12], [18]. The images are 32x32 RGB and are divided into a training set of 50,000 images and a testing set of 10,000 images. The ten classes are uniformly distributed with 6,000 samples each.

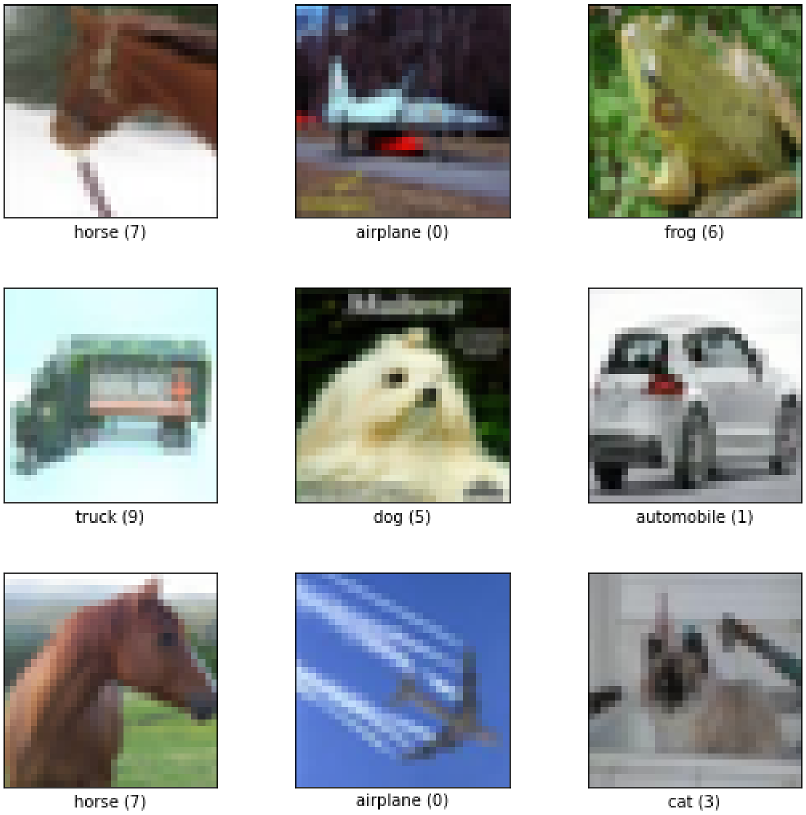


Figure 3.1. CIFAR10 Images and Labels. Source: [2].

3.2.1 Dataset for Centralized Training

For the centralized training, the training set of 50,000 is divided into a training and validation set of 45,000 and 5,000 respectively. The test set of 10,000 is kept together for evaluation after training. Prior to training, the images are converted into float values and divided by 255 to scale the values between 0 and 1. Then the mean of the training data is subtracted from each set of images. During training, the images are padded by 4 zeros, randomly flipped horizontally, and then cropped back to a 32 x 32 sized image. In addition, the class label values are one-hot encoded.

3.2.2 Data Partitions for Federated Learning

The dataset for FL takes on two parts. One that is IID and divided evenly amongst ten clients. This was rather simple to implement, but it took significant work to implement a non-IID data distribution. At first, the two-class per client method of developing a non-IID data partition was explored as seen in McMahan's paper [4]. That required sorting the CIFAR10 dataset into their classes and randomly assigning them to each client. However, other papers explored a more realistic non-IID data partition that used a Dirichlet distribution. The Dirichlet distributions in my experiments have alpha values of 1.0 and 0.5 with the dataset exhibiting more non-IID properties as the alpha approaches zero. The goal of this was to implement a real-world scenario as they generally have non-IID and small datasets [1]. A 2,500-image validation set was set aside for each site to use, giving them the same validation set. This was under the assumption that a server has a known validation set outside of any site's training data. The remaining data were divided evenly across all sites for a total of 4,750 training samples per site. The server uses the test set for evaluation after each federation round. The data augmentation follows the same process for the centralized training except that each site would have a different training mean subtracted based on their subset of data. The server uses the test mean for the subtraction as it remains agnostic of each site's data.

3.3 Neural Network Architecture

As mentioned in Chapter 2, the ResNet was used as the baseline architecture. The deterministic architecture was developed to use a MAP loss function by adding an L2 regularization rate of 0.0001. For the Bayesian models, specific layers were either replaced or added. Tensorflow Probability's flipout layers replaced their corresponding ones in the ResNet architecture. A flipout architecture was chosen because it provides a robust estimation of the posterior distribution during VI [19]. Specifically, the convolutional and dense layers were replaced with flipout versions. The other type of Bayesian NN used in this paper was the MC Dropout [11]. This was achieved by creating a custom dropout layer that would be active during training and inference. This custom dropout layer was added after each convolutional layer and before the final dense layer. Figure 3.2 shows how the original ResNet architecture was modified to create the Bayesian versions. In addition to having a slightly different architecture, the probabilistic models were evaluated using MC sampling, with a sample size of 25. This allowed the uncertainty metrics to be evaluated and the primary ones used in this paper was variance, a measure of epistemic uncertainty.

3.3.1 Flipout Variations

The unique thing about the Flipout Bayesian model is that each weight is a Gaussian distribution with both a mean and standard deviation. With that comes many ways of handling the aggregation of the weights. While FedAvg was adapted to be used with aggregating the means and variances of the distributions between the 10 sites, FedIV was also used to evaluate how different aggregations could affect performance. Another aspect of the Flipout model is that it uses the KLD between the approximation of the posterior and the prior. A naive prior was used for these experiments, meaning that it was a standard normal distribution with a standard deviation of one. In a Flipout model, the KLD is weighted by the number of training samples, but remembering equation 2.7, the standard deviation can be used to affect the prior. By altering the standard deviation of the prior, the model can learn different distributions. As this thesis did not have an ideal prior to start with, the standard prior of $N(0, 1)$ was used along with $N(0, 10)$ to evaluate the results and effects on FL.

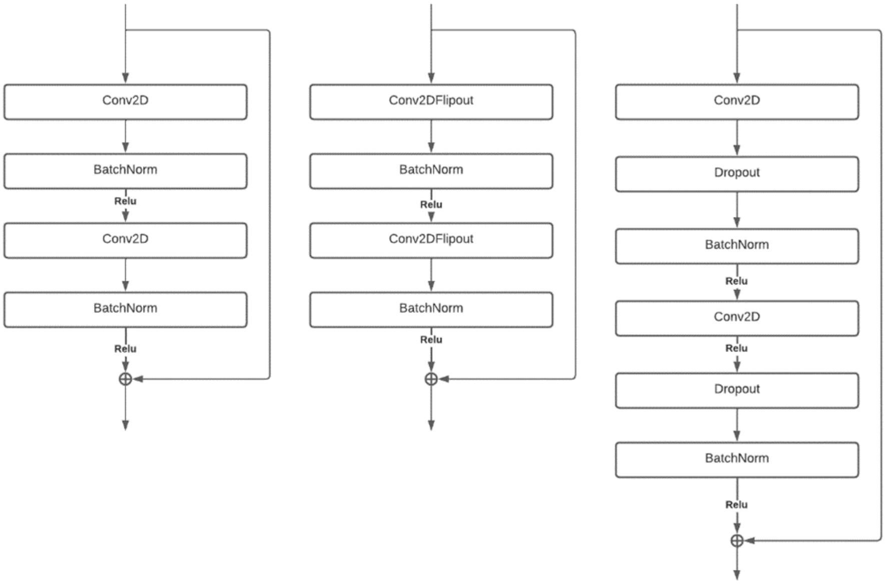


Figure 3.2. Residual Blocks and Variants. Source: [7].

3.4 Experiment Architecture

One of the major products of this thesis was a federated learning framework, utilizing the Flower module, to conduct further research. Flower is a FL framework that has been used to conduct various experiments [20]–[23]. I originally explored implementing Flower in a client-server scenario where each client had one GPU. That limited the number of my clients to three. The way Flower and NVFLARE worked was that a custom client class needed to be developed that received and handled specific arguments from the server during the fit and evaluation phases of training. I implemented a new class for the client which trained on a subset of the CIFAR10 dataset and evaluated their models with VI using MC sampling. Flower implements aggregation through a strategy class. I developed my own strategy class that works for probabilistic weights. Not only did it handle weights represented as a mean and standard deviation, but custom aggregation functions could also be developed and provided to the strategy. For my thesis, I implemented FedIV and FedAvg. After implementing a new strategy class, I investigated different training methods. At first, I approached training as if the server had a pre-trained model which was distributed to the clients. The clients would then train on their local data that was separate from the server's dataset. This was to simulate a continual learning scenario which would be ideal for deployed AI. However, I ended up not pursuing this method since it did not provide an authentic insight into how Bayesian NNs handle a FL scenario comparable to other research. To increase the number of clients and to provide a FL scenario similar to that seen in other papers, I utilized Flower's simulation capability. With the Flower architecture, I conducted simulations of 10 clients with randomly initialized results over the IID and non-IID dataset distributions. Each site would run for one or five epochs on their local dataset and then send their weights back to the server for aggregation. The server would then evaluate the aggregated results on the 10,000-image test set and start the cycle over again. This process is represented in Figure 3.3. This was done for at least 200 federation rounds. These experiments provide a baseline understanding of the immediate impacts of FL to model performance. In addition to this experiment, the models were tested on the IID dataset with sites completing five local epochs before sending their weights back and on two non-IID datasets with varying degrees of distributions.

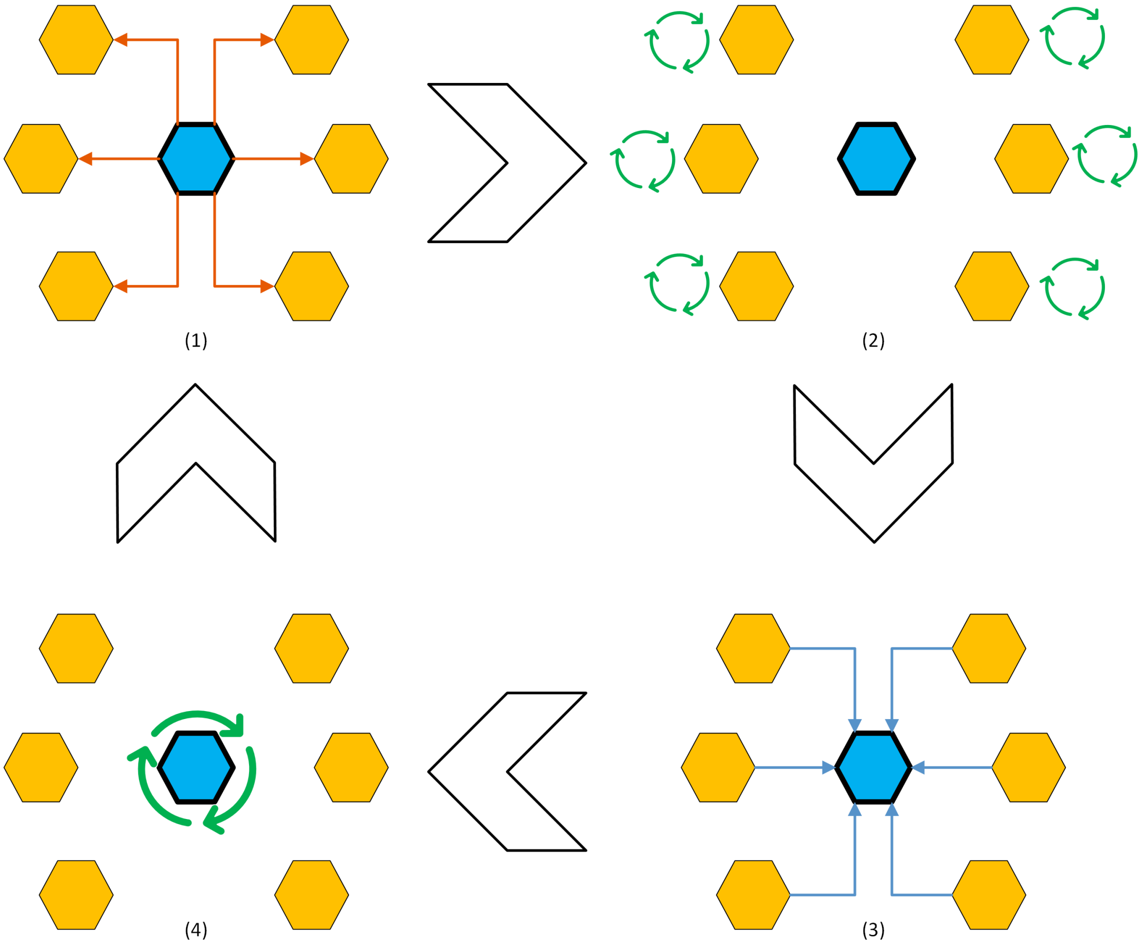


Figure 3.3. The Federated Learning Cycle. 1. Model Distribution 2. Local Training 3. Local Model Updates 4. Server Aggregation

3.4.1 Aggregation Methods

As discussed in Chapter 2, the primary method for aggregation was FedAvg which calculates the weighted average for the global model. Each site's model weights were weighted by the number of samples in their dataset. Since each site received the same amount of data, it becomes a basic averaging formula [12]. For the Flipout models, two different aggregation methods were used, FedAvg and FedIV. The FedAvg function had to be modified in order to take the average of the means and variances separately. In order to average just the means and variances together, functions were developed to create a set of masks for Flipout models.

These masks were provided to the Flower strategy when the model used was Flipout. In addition to FedAvg, I explored a different aggregation function for the Flipout models, FedIV. This weighted the means and variances by their inverse variance and the number of samples [16]. However, it was not my goal to find the perfect method of aggregation but to demonstrate that it could make a significant impact on a model's performance. Depending on the information domain and the purpose of the AI model, certain aggregation methods could be preferred over the standard FedAvg.

3.4.2 Simulation Experiments and Expected Results

Because of limited resources and time, I utilized Flower's simulation function to emulate a real-world operating environment by adding ten clients to the federated learning scenario. This significantly reduces the data partition size of each client as they are only exposed to one-tenth of the CIFAR10 dataset, presenting the challenge faced by many distributed sites in real-world scenarios. The training hyperparameters were kept identical to that of the centralized training. Unfortunately, due to the nature of the simulation, specific callbacks were not able to be utilized and there's a loss of flexibility in the pipeline. This is because, during simulation, the clients are torn down and reconstructed every federation round. This means that the clients lose their previous state. Despite this drawback, these simulations provided a comparison of Bayesian models to their deterministic counterparts in both an IID and non-IID scenario. The first experiment was much like McMahan's et al.'s experiment [4]. It divided the dataset into IID data partitions for ten clients where each client trained for only one epoch before sending back their weights. Like their experiment, I am expecting to see a reduction in performance when compared to the centralized results. For the same data partitions, I am also running a similar experiment except with each client conducting five training epochs before sending their weights back to the server. Potentially, this could overcome the degradation in performance expected from the previous experiment. The next two experiments are on non-IID data partitions in order to see how more non-IID distributions affect performance. These two experiments are much like previous experiments [5], [6] and had clients perform one local training epoch to establish the effect of non-IID data distributions during FL for both Bayesian and non-Bayesian NNs. The expectation is that Bayesian NNs perform as well or better for non-IID data partitions in a FL scenario. The results of these simulations are discussed in the next chapter, Chapter 4.

CHAPTER 4:
Results

The simulations tested on the CIFAR10 dataset are comparable to other works [4]–[6]; however, it expands on this research by comparing state-of-the-art Bayesian and deterministic NNs. By first training the centralized versions of the different types of ResNet20s, an accurate portrayal of how these types of models fare in an FL scenario can be established. My results show how Bayesian NNs that have achieved state-of-the-art results compare to that of deterministic ones. My work also explores the different possibilities that Bayesian NNs bring to the problem set: probabilistic weights and prior distributions. These aspects are addressed through different aggregation functions and by adjusting the prior of a Flipout model. Overall, my thesis contribution provides a foundation for future experiments when it comes to federating a Bayesian NN. The metrics used in analyzing the models and federated learning schemes are accuracy, negative log-likelihood (Eq. 2.2), and variance.

4.1 Centralized Training Results

As previously stated, the centralized training results established a comparison between the deterministic and Bayesian model performances. The goal was to achieve state-of-the-art performance as seen in Kaiming He et al.'s approach [7] for both the deterministic and Bayesian versions of the ResNet20. Each version of the centralized trained ResNet20 achieved results comparable to that in the original paper as seen in Table 4.1, although the Flipout model with a standard prior of 1 was only able to achieve up to 87% accuracy on the test set during my experiments. During training, it was discovered that changing the Flipout prior's standard deviation significantly impacted results. The MC Dropout model and Flipout model with a standard prior of 10 were trained to achieve results slightly exceeding the deterministic version. The prior of 10 was chosen through a grid search and evaluated on the criterion of having the lowest NLL and variance. In addition to the priors, the Flipout with the prior $N(0, 1)$ training pipeline did not include data augmentations. The reasoning was that the strong prior regularized the training so much that there was no additional improvement from the data augmentation. This edit to the training pipeline was also repeated for the Flipout models in the federated learning experiment with the same prior. Figure 4.1 shows the centralized learning curves over accuracy. These learning curves give an idea of how well the different types of models train. The deterministic and MC Dropout models are very similar in their learning curves with the gap between training and validation being smaller for the MC Dropout. This similarity between the training and validation curves could be because of the regularizing effect of the added dropout layers. The figure also shows how different the Flipout models trained compared to every other model, even to each other. These figures will provide a baseline for comparison to the learning curves of the global models in the FL experiments throughout this chapter.

Table 4.1. Centralized Training Results. This table shows the accuracy and variance of the centrally trained ResNet with the lowest NLL. These models were trained with very similar training pipelines except for the Flipout w/ prior std 1. This is due to the severe regularization of this prior and therefore they were not trained on augmented images, however, the normalizing of the pixels remains the same. It is important to emphasize that deterministic model results match benchmark performance reported by He et al. [7] on Cifar10 dataset

Model	Accuracy	NLL	Variance
Deterministic	90.14%	0.462	0.0
MC Dropout	90.91%	0.273	0.0406
Flipout w/ prior std 1	87.21%	0.386	0.0905
Flipout w/ prior std 10	90.42	0.286	0.0474

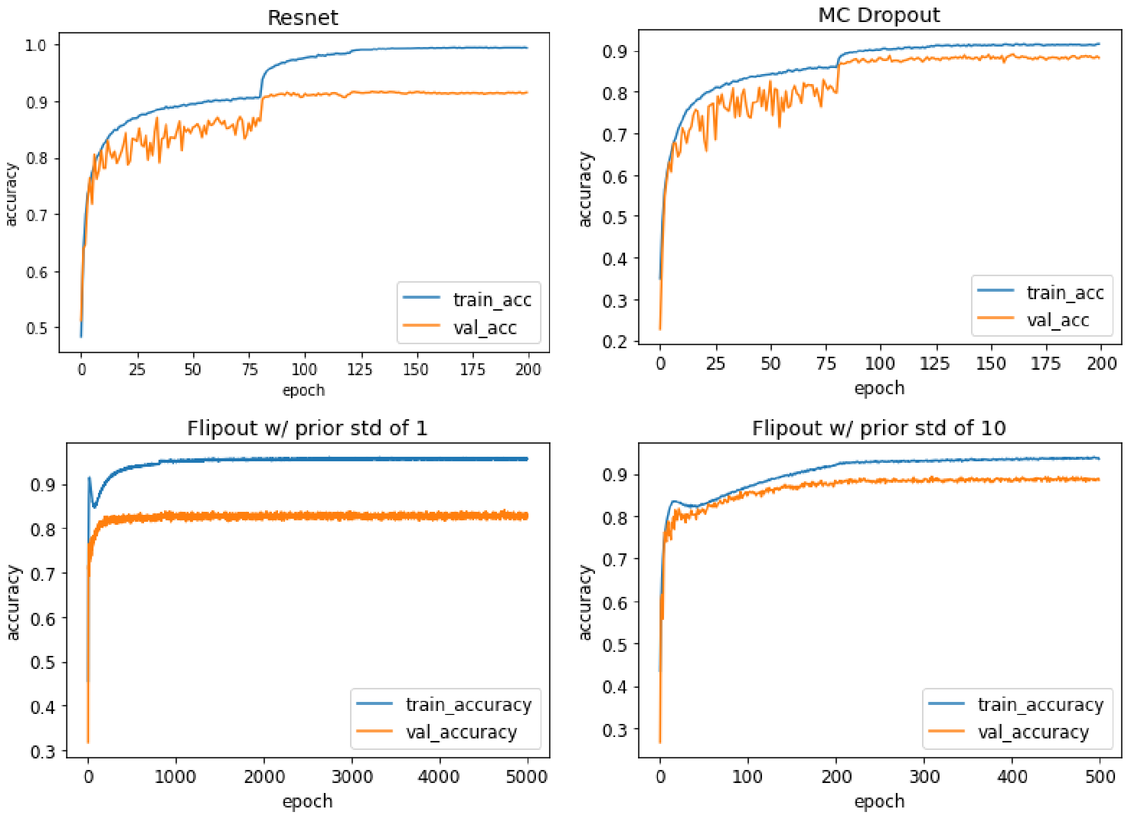


Figure 4.1. Learning Curves over Accuracy for Centrally Trained Models. The Flipout models had priors with different standard deviations. The deterministic and MC Dropout learning curves are nearly identical but with less of a gap between the training and validation curves. Learning rates for that were effective for the ResNet were just as effective for the MC Dropout. For the Flipout models, the one with a stronger prior, prior std of 1, saw little to no improvement after the first hundred epochs. However, a weaker prior shows that the model was able to train to the level of the other types of models.

4.1.1 Bayesian vs Deterministic Models

Before diving into the effects of FL on Bayesian NNs compared to deterministic ones, it is important to understand the difference between them. First is the concept of uncertainty. Although all NNs give a probability value for a given inference, non-Bayesian ones have a poor ability to express uncertainty. Although it can give insight into the aleatoric uncertainty, the deterministic NNs cannot express epistemic uncertainty since its output will always be the same for a given input. On the other hand, a Bayesian model provides a different output each time since it is learning a probability distribution as discussed in Chapter 2. This learned probability distribution allows the Bayesian NN to express its epistemic uncertainty. Expressing uncertainty is a valuable property for applications for which it helps to have high accuracy and uncertainty metrics. As previously discussed, if the Navy applied AI to assess passive sonar, the uncertainty metric could be used to prioritize which samples needed to be reviewed by a human expert. In the health industry, a doctor could prioritize uncertain evaluations to ensure that the AI labeled a patient's condition accurately. Overall, the uncertainty provides a method to understand when the model might be wrong. To demonstrate this difference, example images from the CIFAR10 test dataset were chosen. An example image of a ship and airplane (Figure 4.2) is used to compare the models' predictions to each other.

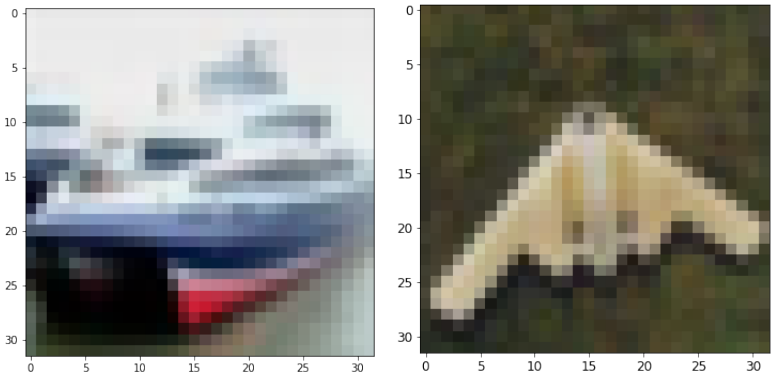
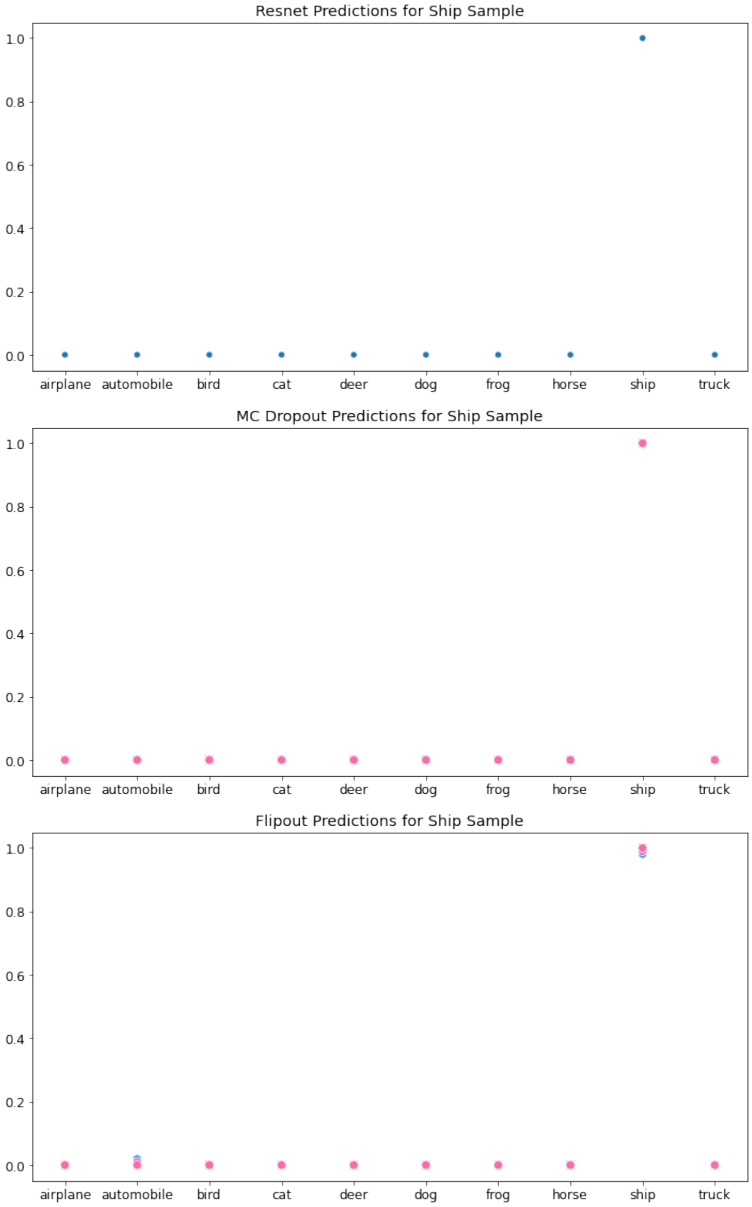


Figure 4.2. Images of a Ship and Airplane.

The first image, the ship, in Figure 4.2, is an example of all three models getting the prediction correct. The second, the airplane, is an example of all three models guessing incorrectly. This section aims to show the stark contrast between the results of all three models. In Figure 4.3, the softmax probability values for the ResNet, MC Dropout, and Flipout, in that order, are plotted for the ship image. Since the Bayesian models are using a Monte Carlo sampling of 25, there are 25 plots on the graphs. In all three cases, the models show high confidence in their predictions with the Bayesian models showing little variance in their sampling. The Flipout model shows a little bit of variance toward the automobile label, which makes sense because a ship and automobile share some properties.


Resnet Predictions for Ship Sample

MC Dropout Predictions for Ship Sample

Flipout Predictions for Ship Sample

Figure 4.3. Predictions on the Ship Sample.

The next image, the airplane, provides a more interesting situation. Each model guessed incorrectly and shows a level of uncertainty for their guesses. The deterministic ResNet shows a bit of uncertainty dispersed between the different labels, but not to the degree to doubt the model's guess. However, in Figure 4.4, the MC Dropout and Flipout form a distribution of predictions with high degrees of variance in two to three classes. The MC Dropout model had guesses that were high in both the airplane, bird, and truck classes. The Flipout model had a wide range in the cat and dog with a few votes for the airplane class. Despite the results being off the mark, it is quite clear that the Bayesian NNs are uncertain in their predictions. They were not as uncertain in the ship example. This section hopes to demonstrate the value that Bayesian NNs bring to a problem.

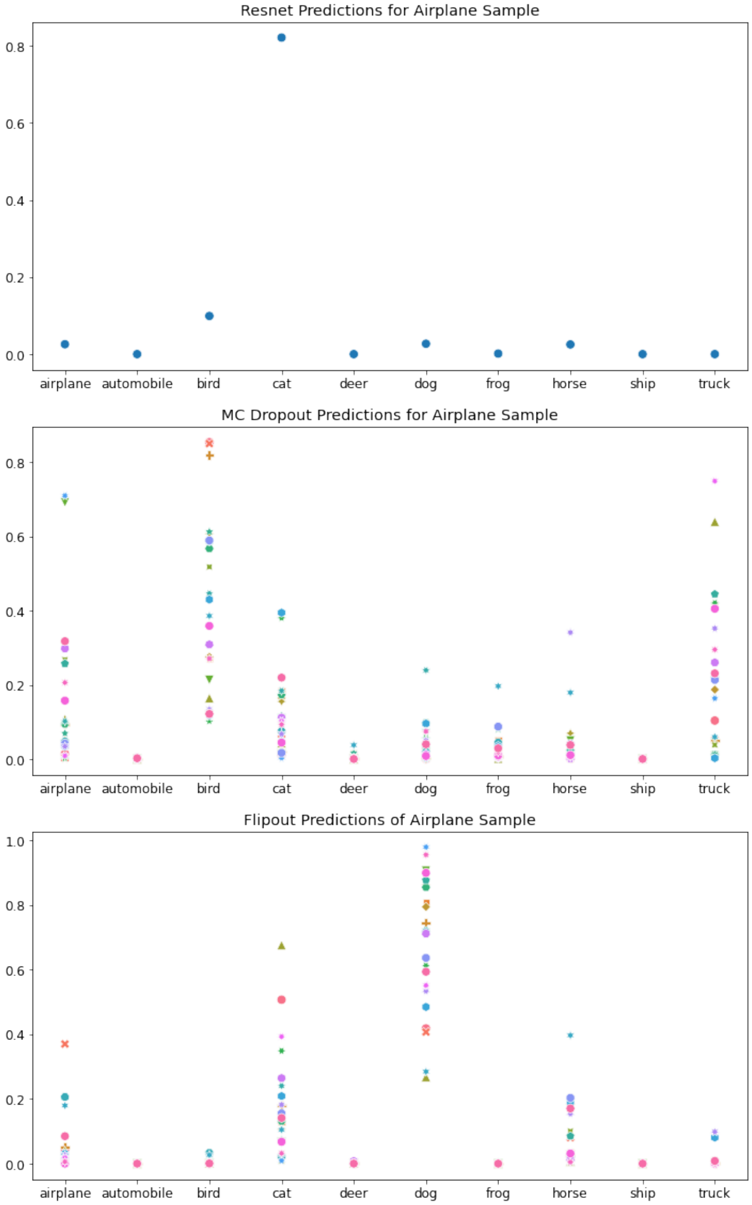


Figure 4.4. Predictions on the Airplane Sample

4.2 Federated Learning – IID Results

This section discusses the results of the federated learning with an IID data partitioning with one and five local epochs per federation round. These models were trained using the methods discussed in Chapter 3. And give a clear understanding of how Bayesian NNs compare to the deterministic NN during FL. The expected results are to see a degradation in performance when compared to the centralized model as seen in McMahan et al's tests on the CIFAR10 dataset [4]. For the increase of local epochs to five, the expectation is that there would be a marked improvement for each of the models.

4.2.1 One Local Epoch

Table 4.2 and Figure 4.5 are the results of the federated learning with IID data partitioning with one local epoch per federation round. As expected, the models experienced a degradation in performance, however, the impact on the Bayesian models was much more significant. With one local epoch per federation round, the deterministic model's accuracy dropped by 2%, but the NLL improved. This suggests that FL could provide a regularizing effect on state-of-the-art architectures and be a way to improve generality in models. The Bayesian models saw a more significant degradation with worse NLLs and a 4% - 8% drop in accuracy. The unique part of this experiment is seen through the Flipout models. For this experiment, the aggregation function made a huge impact on the results. Both Flipout models were able to achieve results comparable to the MC Dropout through FedIV. The FedAvg results showed significant degradation, barely hitting 80% accuracy on the test set. Even though the overall performance decreased for Bayesian NNs, they were more certain in their predictions evidenced by the decrease in variance. Overall, this shows that federating Bayesian NNs is not as easy as federating deterministic ones. The impact of FL has a higher penalty as seen in Table 4.2 and the method of weight aggregation is a significant factor. It is also important to note the learning curves in Figure 4.5. Each model did not see the steps in performance witnessed in Figure 4.1, but there's an insight into how federated models train. The curves for the highest-performing models (ResNet, MC Dropout, Flipout FedIV) are actually very similar. For the Flipout FedAvg models, the learning curves are similar to that in Figure 4.1, but no improvement when the standard prior is one. However, the Flipout FedAvg model with a standard prior of ten looks as if additional federation rounds could see further improvement.

Table 4.2. IID Data Partition Simulation Results - Site Epochs (E = 1). This table shows the accuracy and variance of the global models with the lowest NLL. These global models were trained using 10 clients and used the hyperparameters from the centralized training. Each model was trained for at least 200 federation rounds.

Model	**Agg. Fn.**	**Accuracy**	**NLL**	**Variance**
Deterministic	FedAvg	88.22	0.349	0.0
MC Dropout	FedAvg	86.29	0.416	0.0311
Flipout w/ prior std 1	FedAvg	79.62	0.595	0.0284
	FedIV	87.84	0.361	1.26e-11
Flipout w/ prior std 10	FedAvg	80.61	0.598	0.0718
	FedIV	86.73	0.409	8.66e-12

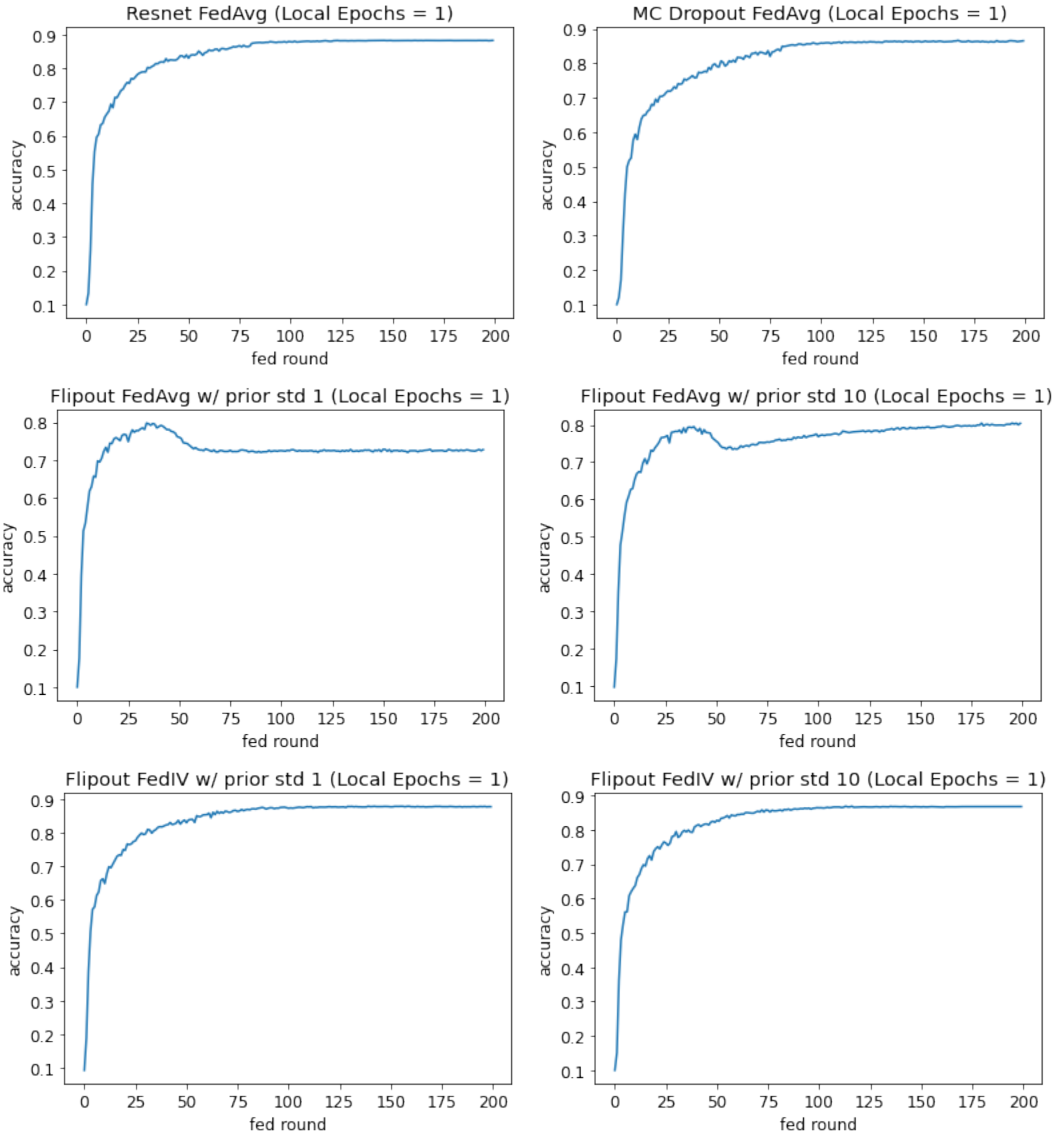


Figure 4.5. Learning Curves of Accuracy for FL on IID Dataset Models with 1 Local Epoch.

4.2.2 Five Local Epochs

Table 4.3 and Figure 4.6 show the models when they are trained for five local epochs at each site before aggregating the model weights at the server. For the majority of the models, this sees a vast improvement with the deterministic model meeting centralized performance and the MC Dropout actually exceeding it with 91.11%. Again, the Flipout models provided an interesting situation as now the model's prior had a significant impact on the results. Contrary to expectations, the Flipout model with a standard prior to one did not improve from the previous experiment. It deteriorated in performance. This could be because with five local epochs, each client's model could be settling on different local minima and when the weights are aggregated, it degrades the overall performance. However, this impact seems to be reduced when the prior is weaker. The Flipout model with the standard prior of ten also improved dramatically from the experiment before with the FedIV and FedAvg algorithms scoring identically. Of note, while the weight aggregation function had a significant impact on the previous experiment, it did not have as big of an impact on this one. When more epochs are involved for each client, the type of model seems more important than the aggregation function.

Table 4.3. IID Data Partition Simulation Results - Site Epochs (E = 5). This table shows the accuracy and variance of the global models with the lowest NLL when trained for 5 epochs at each client. Everything else is as the same as the previous simulation results.

Model	Agg. Fn.	Accuracy	NLL	Variance
Deterministic	FedAvg	90.17	0.333	0.0
MC Dropout	FedAvg	91.11	0.272	0.0390
Flipout w/ prior std 1	FedAvg	73.76	0.829	0.0690
	FedIV	74.23	1.07	6.56e-11
Flipout w/ prior std 10	FedAvg	88.64	0.338	0.0513
	FedIV	88.91	0.389	6.81e-12

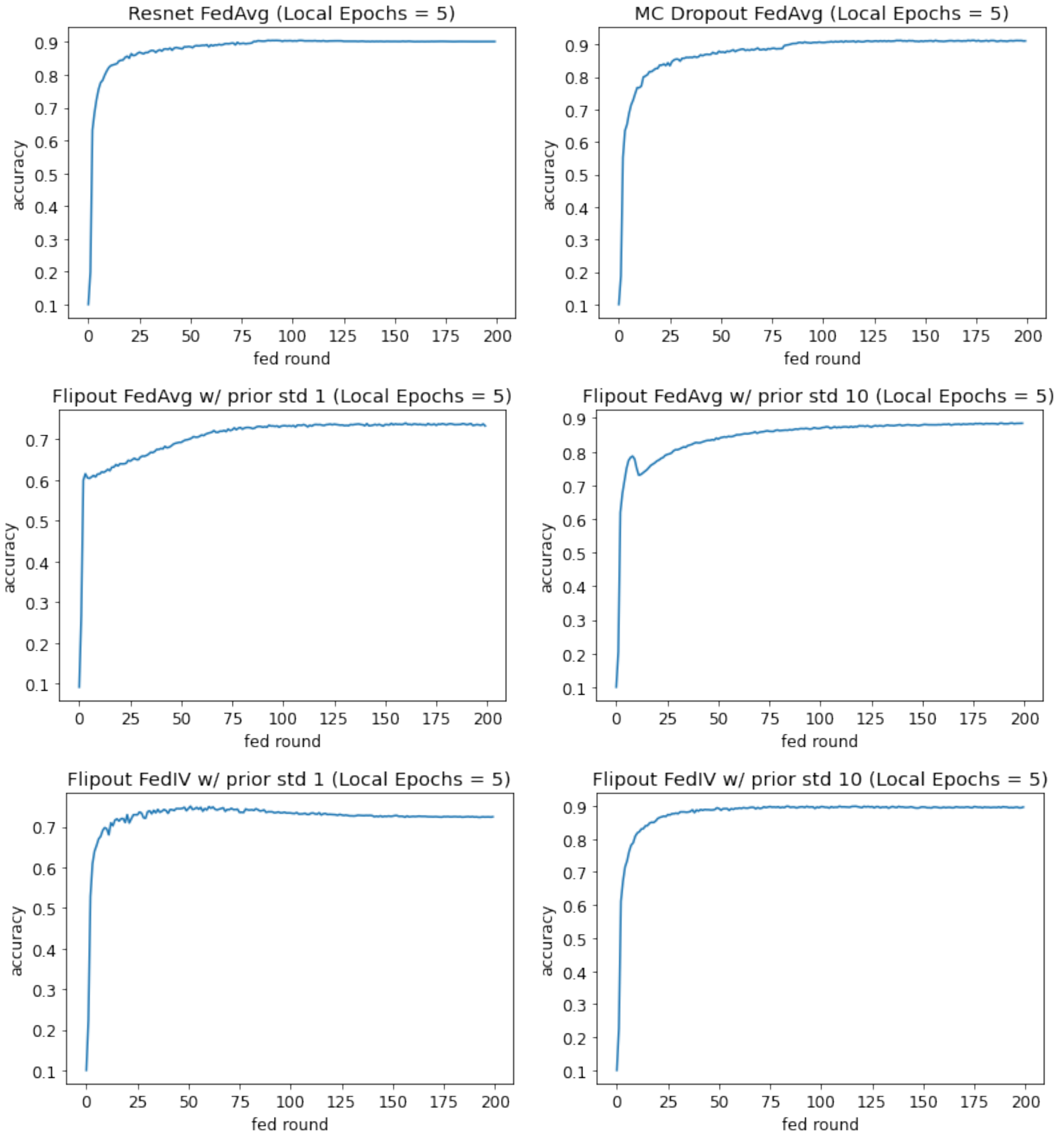


Figure 4.6. Learning Curves of Accuracy for FL on IID Dataset Models with 5 Local Epochs.

4.3 Federated Learning Non-IID Simulation Results

The next set of experiments evaluates how Bayesian NNs perform on non-IID data partitions as compared to the deterministic ones. The expectation was Bayesian NNs would perform as well or better in FL environment. As a review, the non-IID datasets used an alpha value of 1.0 and 0.5 for the Dirichlet distribution. The experiment results are shown in tables 4.4 and 4.5 with their respective learning curves depicted in figures 4.7 and 4.8. When compared to the results in Table 4.2, the models see at least a 4% drop in accuracy and worse NLLs. The results get worse as the data partitions become more non-IID, although the FedIV Flipout with a standard prior of ten and the MC Dropout had the best consistency in performance between the two distributions. Unfortunately, a non-IID data partition had a significant impact on the Flipout models, specifically those with strong priors and/or using FedAvg. Overall, the Flipout models took a 4% to 12% performance drop in accuracy. The worst impact occurred on the Flipout model with the standard prior of one. Unlike in the first experiment, FedIV did not guarantee an improvement in results for the Flipout model. The Flipout model with the stronger prior performed well below baseline for both data distributions with or without FedIV. In contrast, the Flipout model with a standard prior of ten using FedIV performed better than MC Dropout and roughly 8% better than FedAvg. Although a decrease in performance is expected for all the models, the expectation that Bayesian NNs would fare better as well or better than their deterministic counterparts was incorrect. The Flipout models using FedAvg struggled significantly more with non-IID datasets. However, it wasn't just the weight aggregation function that mattered for these datasets, the model itself greatly impacted performance. One observation that was unexpected was that two of the Bayesian NNs had similar performance on both of the non-IID distributions. Bayesian NN's could provide a more robust method of handling operating environments when the local datasets have unknown distributions or even overlapping ones.

Table 4.4. Non-IID Data Partition Simulation Results - alpha = 1.0. This table shows the accuracy and variance of the global models with the lowest NLL when trained for on a non-IID data distribution using the Dirichlet distribution with alpha = 1.0.

Model	Agg. Fn.	Accuracy	NLL	Variance
Deterministic	FedAvg	86.03	0.418	0.0
MC Dropout	FedAvg	82.94	0.515	0.0325
Flipout w/ prior std 1	FedAvg	73.99	0.741	0.0515
	FedIV	72.33	0.803	1.40e-11
Flipout w/ prior std 10	FedAvg	76.75	0.682	0.0287
	FedIV	84.26	0.493	7.58e-11

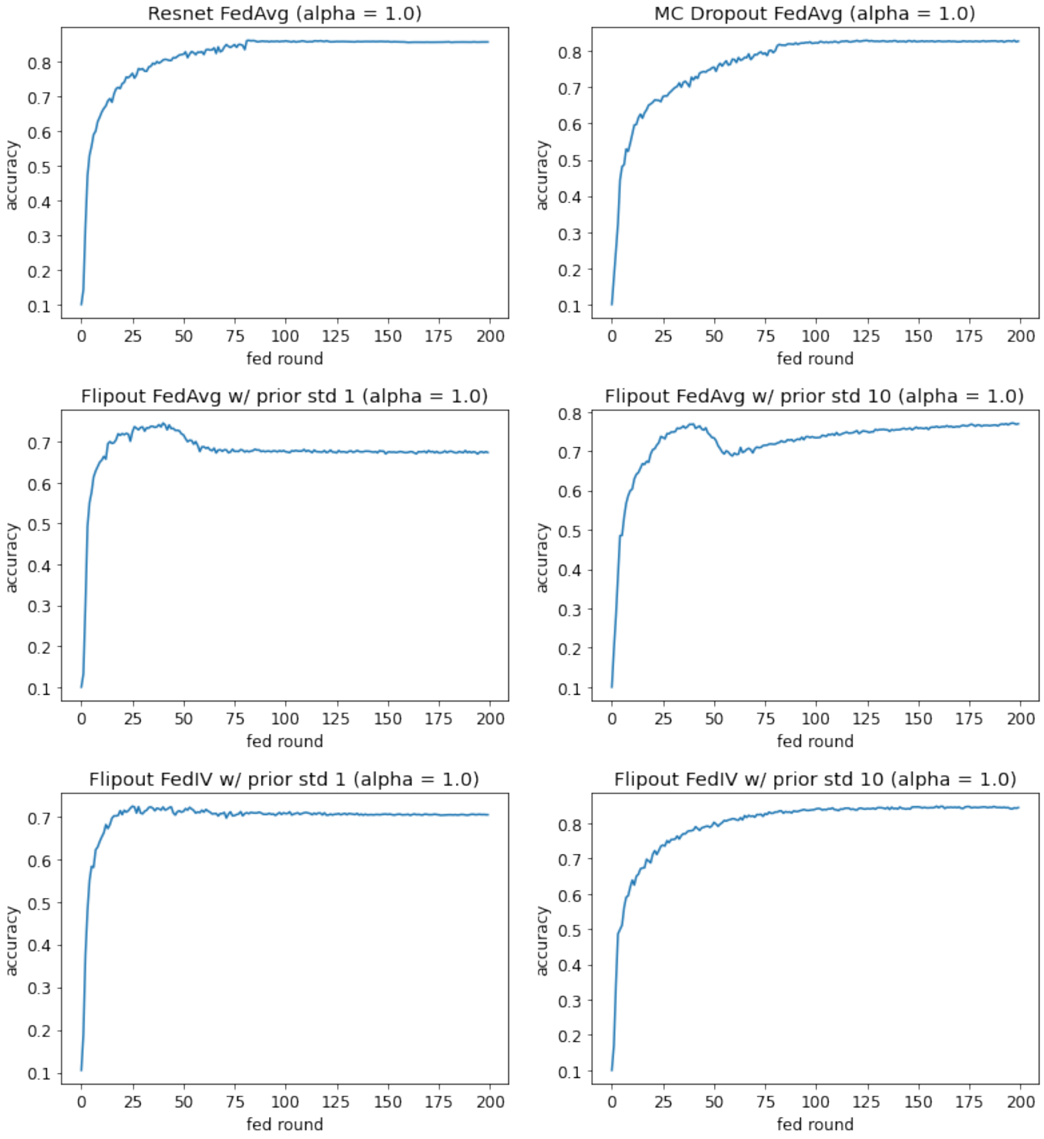
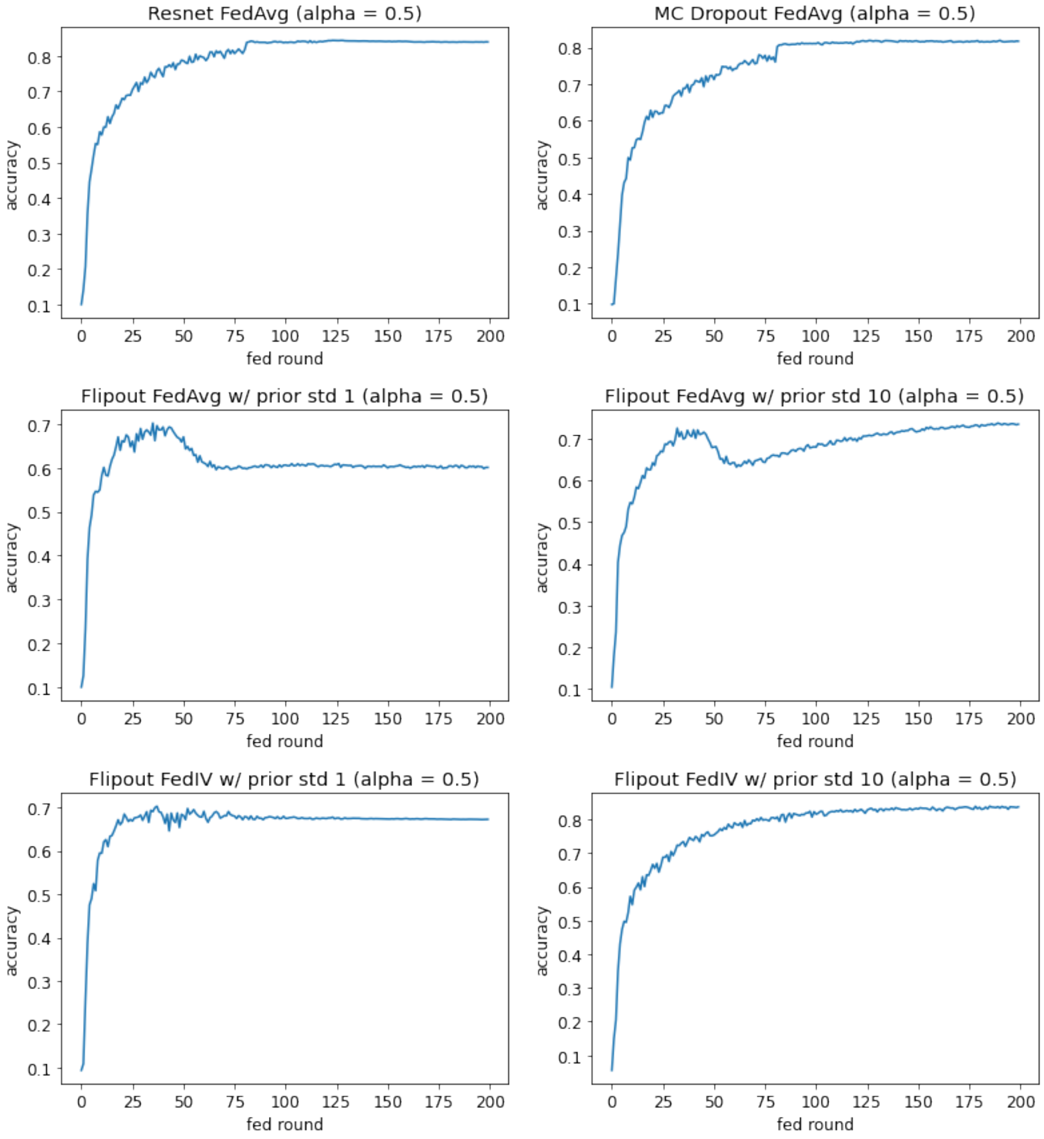


Figure 4.7. Learning Curves for FL Models on Non-IID Dataset (Alpha=1.0).

Table 4.5. Non-IID Data Partition Simulation Results – alpha = 0.5. This table shows the accuracy and variance of the global models with the lowest NLL when trained for on a non-IID data distribution using the Dirichlet distribution with alpha = 0.5.

Model	**Agg. Fn.**	**Accuracy**	**NLL**	**Variance**
Deterministic	FedAvg	84.24	0.462	0.0
MC Dropout	FedAvg	81.77	0.546	0.0323
Flipout w/ prior std 1	FedAvg	68.87	0.850	0.0391
	FedIV	68.44	0.889	1.23e-11
Flipout w/ prior std 10	FedAvg	74.17	0.776	0.0891
	FedIV	83.52	0.515	1.04e-11


Resnet FedAvg (alpha = 0.5)

MC Dropout FedAvg (alpha = 0.5)

Flipout FedAvg w/ prior std 1 (alpha = 0.5)

Flipout FedAvg w/ prior std 10 (alpha = 0.5)

Flipout FedIV w/ prior std 1 (alpha = 0.5)

Flipout FedIV w/ prior std 10 (alpha = 0.5)

Figure 4.8. Learning Curves for FL Models on Non-IID Dataset (Alpha=0.5).

4.4 Analysis of Simulation Results

Over the four different experiments, it can be determined that Bayesian NNs are capable of FL. However, the Bayesian models do not inherently outperform their deterministic counterparts in FL. If a model is going to be naively deployed in an FL situation, the deterministic NN has the best initial results. However, there are a few cases where the Bayesian versions of the models perform as well or better than their deterministic counterparts while providing additional information on uncertainty. In experiment two, MC Dropout was able to achieve results that surpassed the centralized training baseline, while the Flipout models with a standard prior of ten were only a few percentages behind the top performers. Depending on the situation, 2% or 3% could be a valid sacrifice in performance in order to have uncertainty metrics and data privacy offered by federated Bayesian NNs.

4.4.1 Deterministic ResNet

The standard ResNet provided the essential baseline for comparing the Bayesian models. In three out of the four experiments, it had the highest performance in accuracy and NLL demonstrating that Bayesian NNs do not perform as well in FL. One thing of interest is that in the second experiment, shown in Table 4.3, the FL models can achieve state-of-the-art performance. Despite the degradation in results shown in McMahan's experiment with CIFAR10 [4] and observations by Yang et al [1] that weight aggregation does not converge, federated models can, in fact, achieve centralized results. Although further experimentation would have to be conducted for different datasets and state-of-the-art architectures, it is worth noting in of the possibility. With the introduction of FL to a deterministic ResNet, the performance follows similarly with other works on non-IID datasets [4], [6]. Except for the second experiment, FL and non-IID data partitions degraded the deterministic model's performance.

4.4.2 MC Dropout ResNet

In terms of ease of deployment, the MC Dropout was the easiest Bayesian NN to implement. Due to its inherent properties, I did not have to modify Flower's FL architecture in order to deploy and aggregate MC Dropout models. In terms of performance, the MC Dropout performed closely to the deterministic counterpart as witnessed by all of the learning curve figures. In all the IID experiments, the MC Dropout performed closely to the deterministic version in FL, actually achieving the top performance when the sites performed five local epochs per federation round. This could be because, during training, a random neuron is dropped as discussed in Chapter 2. With more iterations, there's a wider distribution of neurons that have been trained, improving the model overall. Also, in experiment one, the NLL improved for the deterministic model. Usually, an improvement in NLL equates to an improvement in the generality of a model. Although the MC Dropout did not have this effect in the first experiment, it could have occurred in the second. Therefore this dropout property when paired with FL seems to complement one another, allowing the global model to achieve greater than its baseline results in the second experiment. These properties seem to scale well with additional training epochs at the site level. In terms of the non-IID data partitions, it had one of the lowest change in performance between the two Dirichlet distributions. This could mean that MC Dropout in federated learning handles varying data distributions well, making it a valuable model when data distributions at each site are not guaranteed to be IID.

4.4.3 Flipout ResNet – FedIV vs FedAvg

The Flipout ResNet provides the most unique and complex situation with the FL scheme. As the weights are the mean and standard deviation of a Gaussian distribution, there are many ways to interpret how to aggregate the weights. The following compares the inverse variance aggregation method, FedIV, and the averaging of random variables discussed in Chapter 2 via FedAvg. FedIV, in many situations, dramatically improved results. The models trained with FedIV performed closest to the deterministic model and the centrally trained version in terms of accuracy and NLL on the IID datasets. The improvement from FedIV is effective when trained for one local epoch at the site as the improvement is negligible for the Flipout models when the local epochs are increased to five per federation round. This could be because each site trains on a different dataset, they reach different local minima and when the local minima are too far apart, the aggregation lands somewhere in between. When aggregated, the weights might not even fall on a local minimum anymore contributing to the degradation in performance as seen in the second experiment. For non-IID data partitions, the FedIV actually improved results when compared to FedAvg, at least for the Flipout models with a standard prior of ten. It even had the lowest change in performance between the two data distributions. This shows that Flipout models with FedIV can perform consistently well on data partitions with varying degrees of non-IID distributions. Overall, FedIV improved results for the Flipout model. The models were more certain in their inferences which could be due to the variance-reducing properties of the inverse variance algorithm [16]. When paired with the Flipout model with a standard prior of ten, the FedIV aggregation function outperformed FedAvg in every experiment. More experimentation would have to be done for different datasets, but FedIV appears to be a viable improvement to FedAvg. In any case, this experiment shows that the type of weight aggregation function significantly impacts results.

4.4.4 Flipout ResNet – Strong vs Weak Priors

Another aspect of the Flipout model that significantly impacted results in both a centralized and FL setting was the prior. When developing a Flipout model with Tensorflow-Probability, the Flipout layers are initialized with a standard prior set around one. During my development of a Flipout ResNet, I struggled to find the right training pipeline to achieve state-of-the-art performance as seen in Table 4.1. In general, the Flipout models saw a significant degradation in performance when compared to the MC Dropout and deterministic versions in both a centralized and FL setting. I assess that this is an impact of how the loss is calculated for a Flipout model. In equation 2.18, the KLD is taken with respect to the model's learned distribution and the prior. In practice, the first term in equation 2.18 is averaged by the number number of its training samples, meaning that the second term, the KLD, is also averaged by the same. In an FL setting, the number of training samples was reduced by a factor of ten, meaning that the KLD term in equation 2.18 is ten times stronger for each client. This means that the Flipout model at each client would prioritize being close to the prior versus learning the dataset. This is even further exacerbated by the strength of the prior. When the prior has a small standard deviation, let's say one, then the regularization factor in the KLD is approximately 0.5. Most L2 regularization is set to 0.0001 by default. This means that a standard prior of one is potentially 5,000 more regularized than a standard model. To mitigate this, the prior was increased to ten to achieve a regularization rate closer to standard practice. These two factors presented by a FL scenario and the model itself could explain why the Flipout model achieved sub-par results. These observations are further supported by the fact that the Flipout model with a standard prior of ten consistently outperformed the other in both centralized learning and FL. For instance, in figures 4.5, 4.7, and 4.8, the strong prior achieved optimal performance early and then did not improve. On the other hand, the weaker prior had the same trend but did improve with additional federation rounds. Exploring the effects of a prior for on Bayesian NNs is essential to understand how to deploy them in a FL environment.

4.4.5 Overall

The results from these experiments demonstrate that deploying Bayesian NNs in a FL setting is not as easy as the deterministic NNs. In a perfect data environment where the dataset is IID, Bayesian models can achieve state-of-the-art while providing metrics of uncertainty. Although they perform worse in a non-IID FL scenario, the distribution factor did not make a significant impact on the results. The MC Dropout and FedIV Flipout model (w/ prior std 10) had the best consistency between the two non-IID distributions, a valuable property. Perhaps by increasing the number of epochs for non-IID data partitions, Bayesian NNs could achieve results similar to an IID dataset. Another alternative would be deploying a different weight aggregation function that could handle non-IID data partitions better. These experiments conclude that the operating and training environments for Bayesian models deeply impact the results of FL and must be taken into account. It also shows that Bayesian NNs are capable of handling varying degrees of non-IID data partitions while providing quality uncertainty metrics.

CHAPTER 5:
Conclusion

5.1 Contributions

The main contribution of my thesis was an evaluation of benchmark quality CNN models in both deterministic and Bayesian configurations with FL. Second, I expanded a well-established framework with the ability to handle Bayesian CNNs with mathematically correct aggregations of probabilistic weights. After exploring NVFLARE and Flower python packages, I settled on using Flower in order to run simulations run on the CIFAR10 dataset using 10 clients. The objective of this was to be able to produce repeatable results even when using FL platforms. For example, if the Navy or DoD decided to use NVIDIA hardware, Flower is supporting NVIDIA embedded boards [20]. In addition to providing a FL framework, I also explored how different types of Bayesian NNs were affected by FL. I created different classes and functions that made it possible to federate a Bayesian NN like Flipout. My framework also has the capability of being expanded on with different datasets, callbacks, clients, and NN architectures for further research discussed later in this chapter.

With the federation of Bayesian NNs there are some key insights that my thesis contributes.

1. Bayesian NNs can be utilized in a federated setting and achieve state-of-the-art results like MC Dropout scoring 91%.
2. Weight aggregation function has a significant impact on results. In most cases, the inverse variance aggregation method, FedIV, improved the Flipout model's results. It improved results ranging from 6% - 9%, depending on the experiment.
3. The model's prior also has a significant impact on Bayesian NNs in both centralized training and FL. This could be due to clients having smaller dataset sizes, which means the KLD term in the loss function (equation 2.18 is not as heavily weighted during FL. It is possible that during FL the models tended to prioritize reducing the KLD term instead of the NLL.
4. Bayesian NNs can handle non-IID data distributions consistently. The MC Dropout and FedIV Flipout (w prior std 10) had a 1% change in accuracy when the Dirichlet

distribution alpha was halved. However, further research is required to improve the overall performance of Bayesian NNs on non-IID data distributions.

5.2 Future Work

Although this thesis provided a framework for testing Bayesian NNs, it does not do a comprehensive study of how to optimize FL for them. Further studies on the Bayesian models with weights expressed as Gaussian distributions are necessary to determine what kind of aggregation functions work best. There's not a lot of consensus on how a Gaussian distribution should be averaged and each method has its statistical reasoning. It could be dependent on the dataset or the needs of the user. Either way, more studies need to be conducted. Another area of work would be to repeat Hsu et al. work on the effects of non-IID datasets on FL [6] but with Bayesian NNs. By investigating how the distributions affect a Bayesian NN in FL, it could validate or invalidate my claim that Bayesian NNs are more consistent with non-IID datasets. In addition to these effects, there needs to be further work on combating the negative effects of non-IID and small datasets. This would further help the Navy and DoD since it would require less time, energy, and taxpayer dollars to build a dataset if it didn't have to be IID. Further work could also be done on different datasets. This experiment only explored one dataset, CIFAR10. Other experiments could be done on CIFAR100 or other benchmark datasets. This would validate any trends seen in my thesis and provide a collective insight into how the data domain affects a federated Bayesian NN's performance. Other datasets could be passive sonar or drone imagery for applications in the Navy and DoD. When it comes to deploying AI for the Navy and private industries, transfer learning cannot be overlooked. My experiment could also be adapted to conduct FTL or even continual learning. By doing weight aggregation on a subset of a NN's layers, FTL can be accomplished, but the effects of FL on transfer learning are not clear, especially for Bayesian NNs. Lastly, my experiments were conducted using a simulation of ten clients. Initially, my thesis was meant to federate edge devices with Bayesian NNs, but it now falls into the category of future work. The next step would be to implement various edge devices in a network that conducts FL. Noting the hardware impacts of FL, such as network, CPU, and electricity usage is important to understand for edge devices. This would be the prototype stage for Navy and DoD applications. Using this framework for sonar buoys or drones can demonstrate the real-life importance of this kind of research.

5.3 Final Remarks

To summarize, this thesis met its goal of federating Bayesian NNs. It provides the necessary framework to apply FL to other areas of research. By providing the necessary datasets and NNs, this experiment can quickly be adapted to a specific problem set. I believe FL has the potential to effectively deploy AI on the edge for the Navy and DoD. When paired with Bayesian NNs, it has the capability of continual learning and removing the need to have a large database. This is advantageous when deploying AI across commands, services, and even agencies. The potential for AI to be on the edge, can also improve warfighting capabilities by reducing the need for communications. If the AI was Bayesian, it could provide even more information through its ability to express uncertainty. Decisions need to stay with decision-makers, but it always helps to have cutting-edge technology and information and to know when that information or system could be wrong. To conclude, FL could be the most ideal way of deploying effective and continually growing AI for the Navy and the rest of the DoD.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] Q. Yang, *Federated Learning* (Synthesis lectures on artificial intelligence and machine learning, #43). San Rafael, California, USA: Morgan & Claypool, 2020.

[2] A. Krizhevsky *et al.*, “Learning multiple layers of features from tiny images,” 2009 [Online] http://www.cs.utoronto.ca/ kriz/learning-features-2009-TR.pdf.

[3] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, “Federated optimization: Distributed machine learning for on-device intelligence,” *CoRR*, vol. abs/1610.02527, 2016 [Online]. Available: http://arxiv.org/abs/1610.02527

[4] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *CoRR*, vol. abs/1610.05492, 2016 [Online]. Available: http://arxiv.org/abs/1610.05492

[5] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, T. N. Hoang, and Y. Khazaeni, “Bayesian nonparametric federated learning of neural networks,” 2019 [Online]. Available: https://arxiv.org/abs/1905.12022

[6] T.-M. H. Hsu, H. Qi, and M. Brown, “Measuring the effects of non-identical data distribution for federated visual classification,” 2019 [Online]. Available: https://arxiv.org/abs/1909.06335

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[9] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training very deep networks,” *Advances in neural information processing systems*, vol. 28, 2015 [Online]. Available: https://doi.org/10.48550/arXiv.1507.06228

[10] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” *arXiv preprint arXiv:1505.00387*, 2015 [Online]. Available: https://doi.org/10.48550/arXiv.1505.00387

[11] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” 2015 [Online]. Available: https://arxiv.org/abs/1506.02142

[12] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy preserving machine learning," Cryptology ePrint Archive, Paper 2017/281, 2017, https://eprint.iacr.org/2017/281. Available: https://eprint.iacr.org/2017/281

[13] K. V. Sarma, S. Harmon, T. Sanford, H. R. Roth, Z. Xu, J. Tetreault, D. Xu, M. G. Flores, A. G. Raman, R. Kulkarni *et al.*, "Federated learning improves site performance in multicenter deep learning without data sharing," *Journal of the American Medical Informatics Association*, vol. 28, no. 6, pp. 1259–1264, 2021 [Online]. Available: http://dx.doi.org/10.1093/jamia/ocaa341

[14] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip learning as a decentralized alternative to federated learning," in *Distributed Applications and Interoperable Systems*, J. Pereira and L. Ricci, Eds. Cham: Springer International Publishing, 2019, pp. 74–90.

[15] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011 [Online]. Available: https://ieeexplore.ieee.org/document/5640675

[16] W. G. Cochran and S. P. Carroll, "A sampling investigation of the efficiency of weighting inversely as the estimated variance," *Biometrics*, vol. 9, no. 4, pp. 447–459, 1953. Available: http://www.jstor.org/stable/3001436

[17] M. Orescanin, M. Ergezer, G. Singh, and M. Baxter, "Federated fine-tuning performance on edge devices," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2021, pp. 1174–1181.

[18] C. He, A. D. Shah, Z. Tang, D. F. N. Sivashunmugam, K. Bhogaraju, M. Shimpi, L. Shen, X. Chu, M. Soltanolkotabi, and S. Avestimehr, "Fedcv: A federated learning framework for diverse computer vision tasks," *arXiv preprint arXiv:2111.11066*, 2021.

[19] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. B. Grosse, "Flipout: Efficient pseudo-independent weight perturbations on mini-batches," *CoRR*, vol. abs/1803.04386, 2018 [Online]. Available: http://arxiv.org/abs/1803.04386

[20] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning framework," 2022 [Online]. Available: https://doi.org/10.48550/arXiv.2007.14390

[21] X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou, "Fedbn: Federated learning on non-iid features via local batch normalization," *arXiv preprint arXiv:2102.07623*, 2021 [Online]. Available: https://doi.org/10.48550/arXiv.2102.07623

[22] W. Zhao, X. Qiu, J. Fernandez-Marques, P. P. de Gusmão, and N. D. Lane, “Protea: Client profiling within federated systems using flower,” in *Proceedings of the 1st ACM Workshop on Data Privacy and Federated Learning Technologies for Mobile Edge Network*, 2022 [Online], pp. 1–6. Available: https://doi.org/10.48550/arXiv.2207.01053

[23] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, “Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 876–12 889, 2021 [Online]. Available: https://doi.org/10.48550/arXiv.2102.13451

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California